The Dissertation Committee for Brandon Keith Wiley Certifies that this is the approved version of the following dissertation:

Circumventing Network Filtering with Polymorphic Protocol Shapeshifting

Committee:

William Aspray, Supervisor

Kenneth Fleischmann

Byron Wallace

Jacek Gwizdka

Margaret Myers

George Danezis

Circumventing Network Filtering with Polymorphic Protocol Shapeshifting

by

Brandon Keith Wiley, B.A., B.S., M.A.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2016

Dedication

I dedicate my dissertation to my parents and my sister.

Acknowledgements

I wish to thank my committee members. A special thanks to William Aspray, my committee chairman, for encouraging me to choose a topic close to my heart for my dissertation, and Maggie Myers, the committee member that has stayed with me from the beginning of the dissertation process when it was just an idea.

Circumventing Network Filtering with Polymorphic Protocol Shapeshifting

Brandon Keith Wiley, PhD

The University of Texas at Austin, 2016

Supervisor: William Aspray

As use of the Internet has expanded to become ubiquitous, so has the use of filtering technology to selectively block access to content. In order to restore access to blocked content, filtering circumvention technologies have also been developed. This dissertation addresses the technical methods for developing a circumvention technology that is both useful and pragmatic. This goal was achieved by using modeling of filters as the basis for constructing filtering-resistant encodings that provide adaptability to filtering methods. The result was an engine that can generate encodings customized to work with different circumvention tools and to circumvent different filters. Through a value-sensitive design process a circumvention tool was developed that utilizes this engine to provide access to credible and relevant information for users on networks that implement filtering. This research has implications for the practical effectiveness of filtering technologies and the methodologies for the creation of circumvention tools.

Table of Contents

List of Tables

List of Figures

Figure 22: Accuracy of incoming entropy classifier for first iteration of test adversary 80

Figure 25: Accuracy of incoming length classifier for test adversary iteration 1..82

List of Illustrations

1. Introduction

Since the creation of the Internet, a tension has existed between the desire of users to access credible and relevant information and the policies of network administrators that are created with the aim of regulating content. As new Internet-based communication technologies are developed, they are used to further the goals of each side. A cat and mouse game has followed in which new a filtering technology is developed, then a new circumvention technology is created to bypass the filtering. As network speed and computing power have increased, ever more sophisticated technologies have been used for both filtering and circumvention.

This back and forth has provided fertile ground for network security researchers, with the research agendas following the trends in the deployment of filtering and circumvention methods. In the first wave of research, censorship-resistant publication systems were developed in response to attacks on websites hosting specific content that the adversary wanted removed [4]. In the second wave, filtering moved to blocking access to the sites holding the content rather than removing the content itself, leading to approaches that are used to classify and block network connections, such as *shallow packet inspection*. As a response, circumvention technology was developed that hid the aspects of network communication that were used to filter network connections to blocked sites [5]. In the third wave, now underway, techniques such as *Deep Packet Inspection (DPI)* are being used to classify and block network connections based on what protocol is being used, rather than what site is being accessed. This approach has led to the current wave of circumvention research, which is concerned with developing obfuscating protocols that the *DPI* technology cannot classify correctly [6]. This technology is referred to in the literature as network protocol obfuscation or more generally as unobservable communication.

Unobservable communication is a sociotechnical problem that requires a precise and pragmatic definition of the observer. Research in this field takes distinctly different directions, depending on whether the proposed observer is chosen based on theoretical or practical concerns. One approach attempts to define the limits of what can be made unobservable given an observer with very strong properties taken from the literature [18], while the other approach attempts to make tools that are both usable and effective [9]. Each of these approaches can be found within the body of Computer Science literature where most work on unobservable communication resides. The research that embraces the theoretical approach has well-defined threat models together with strong results for those models; but when the threat models do not match the reality of filtering conditions, the results lack practical relevance for the developers and users of circumvention tools. The research that takes the practical approach makes relevance to the needs of users the primary goal. However, when the threat models are not well-defined, the tools can fail to be effective. When the tools are used against filters that provide a different set of threats, they are unlikely to succeed. The research described here synthesizes aspects of both the practical and theoretical approaches by using well-defined threat models that are also practical. The threat models were based on observations of filtering hardware and analysis of filtered networks. This approach is both a synthesis and an advancement of the theoretical and practical approaches. It uses a threat modeling approach found in the theoretical research, but also builds realistic threat models that are useful in the problems studied by the practical approach.

The field of Information Studies is concerned with the sociotechnical triangle of interactions between humans, information, and technology. When looking at the problem of unobservable communication from the perspective of Information Studies, the human component is of critical importance in differentiating this dissertation research from related research in Computer Science. A focus of this dissertation is therefore how this research will benefit people. The circumvention technology developed through this research has not been treated as politically and socially neutral, but as a value-laden construction that encodes the values of the situation and purpose for which it is being constructed. Rather than taking a general-purpose approach to unblock all filtered Internet traffic, the circumvention techniques developed here has been created with the aim of enabling a specialized use case for a specific user community. The lasting value of this research is the generalizable method of creating specialized tools for communities.

1.1 PURPOSE

In the modern Internet, filtering is prevalent and is used for a variety of reasons from blocking of malware to political oppression. While in the past Internet traffic was filtered based on IP addresses visited or specific content viewed, currently the dominant form of filtering is protocol classification. Existential and statistical properties of the traffic are analyzed to determine which protocol is being used. Based on the protocol, the traffic is classified into one of a fixed set of categories. The categories are provided in a high-level interface to network administrators who can, with the click of a button, block all traffic within a category. This approach blocks all network connections using a specific protocol, regardless of the content being accessed. Accurate filtering relies on the ability of the filtering hardware to classify the traffic into protocols by observing properties of the filtered traffic.

In order to use a value-sensitive approach for designing a circumvention method, a set of values must be chosen. These values have been selected to inform questions related to topics that are of interest to Information Studies. The topics chosen to inspire the value set are the following:

- Information access
- Information credibility
- User relevance

Based on these topics of interest, the core value chosen for the design is providing access to credible and relevant information. There are a multitude of uses for network filtering and some of them, such as malware blocking and intrusion detection, do not conflict with this value; so circumvention of these uses is not necessary for this design. However, filtering can also be used to block access to relevant and credible information. For example, the most extensively filtered content on the Internet is news. On many networks where information policy is set at the national level, access to credible news sources is blocked. This censorship of news media can extend beyond major news sites to individual blogs and even to individual news stories and blog posts. The control of access to relevant and credible information can extend beyond the targeted sites to any protocols that can be used to bypass the blockade on news. Virtual Private Network (VPN) software that is used by business travelers to securely connect to their home networks is now blocked on some networks that block access to news. Since encrypted protocols are sometimes used to circumvent filtering, encrypted protocols are also a target for blocking. Even protocols such as HTTPS, which are critical to providing security on the web, are sometimes blocked. Extreme efforts to filter news can therefore, through collateral damage, be at odds with supporting security and privacy - the same benefits that network filtering devices could theoretically provide if used for blocking malware and other attacks.

The purpose of the proposed dissertation project is to support the core value of allowing access to credible and relevant information by designing and implementing a new filtering circumvention technology that renders protocol classification ineffective as a means of filtering.

This research has the following goals:

- **Advance the literature of unobservable communication** - Show that the approach used for building circumvention technology is a general solution to the problem of filtering based on protocol classification
- **Practical** - Demonstrate that the circumvention technology is effective against sociotechnically realistic scenarios
- **Useful** - Explicate the ramifications of this technology on the specific use case of providing access to credible and relevant information

1.2 Research Questions

Modeling Filters

- What properties of Internet traffic are important for filtering as it occurs in practice today?
- Do the particular filtering methods that have been implemented in deployed hardware have a set of characteristics that provide an opening for a practice of circumvention?
- How can statistical models be used to capture the relevant details of filters?
- How well do the statistical models of protocols fit the observed data?

Design of Circumvention Tools

- How can a circumvention tool be built using models of filters?
- How effective is the circumvention tool against the modeled filters?
- What are the characteristics of traffic carrying credible and relevant information that allow it to be classified and filtered?
- How effective can a circumvention tool be in restoring access to this information when evaluated against a simulated filter?
- How efficient can a circumvention tool be in restoring access to this information in terms of bandwidth overhead?

1.3 Importance

As new filtering technologies have been developed, new methods of circumvention have been introduced. Effective circumvention tools then necessitate new filtering technologies, and so on, in a cycle. Current approaches to circumventing modern filtering techniques have followed a similar approach to circumventing previous filtering techniques. New protocols are developed that circumvent current filters, but they last only a single technological generation before new filtering technology is developed to defeat the circumvention tools.

The research proposed here breaks out of this cycle by taking a sociotechnical and information-theoretic approach to analyzing the problem of filtering. The general-purpose approach to circumventing filtering based on protocol classification was adapted into a pragmatic instantiation that is effective at circumventing current filters. If filtering techniques or policies change, the process of instantiating a specific circumvention tool for the current conditions can be repeated. This approach turns what was previously a technology development problem into an information problem. This shift breaks the technological chess game between filtering and circumvention and moves into a new direction where the competition is not between software or hardware but between the fidelity of models. Specifically, the competition was between whether the filtering hardware or the circumvention tool had a better computational model in terms of most accurately reflecting the sociotechnical landscape of Internet use. The goal of the filter was to define in precise computational terms the observable difference between types of Internet use that the network operator desires to allow and those the operator desires to block. The goal of the circumvention tool was to make the line between these two categories difficult to determine with precision. In information-theoretic terms, the goal is to make these two categories practically indistinguishable for the given observer. This approach of combining a sociotechnical perspective with an information-theoretic analysis of filtering and circumvention is currently not employed in the field of unobservable communication. By applying an Information Studies perspective to a field that is usually only studied from a purely technical point of view, new results are possible that enable unobservable communication against specific filters to be both effective and efficient.

This research is important not only because it advances the literature of Information Studies and of unobservable communication, but also because it provides a practical and useful means to advance the core value of the design. As the Internet has become an increasingly important source for satisfying everyday information needs for people around the world, new threats to information access have arisen. Unlike physical media, access to online information can be revoked instantly on a national scale. Through an appropriate circumvention tool access to credible and relevant information can be restored effectively and efficiently for a user community affected by filtering.

1.4 Approaches to Filtering and Circumvention

This introduction will explore some of the history of filtering and circumvention. The literature review is kept intentionally narrow, as the purpose is to establish the motivation for engaging in the present research, which is the development of *polymorphic protocol shapeshifting* systems as a means to circumvent network filters. Once this course of research has been established and the ideas put forth in this dissertation are accepted by the research community as a promising approach, there is additional literature that could be explored to build improved systems. Literature from steganography, website fingerprinting, and machine learning could all provide potential future improvements upon the system presented here. Additionally, research on protocol obfuscation continues to evolve as new systems are created. This literature review does not attempt to provide a comprehensive overview of every obfuscation method, but rather to focus on the approaches that precede and motivate the present work.

The evolution of filtering on the Internet can be characterized as a cycle of increasingly sophisticated filtering techniques inspiring new circumvention techniques, leading to the development of more advanced filtering techniques. *Shallow packet* *inspection* refers to an approach where only packet headers are examined. *Shallow packet inspection* filtering techniques that filter based on the IP addresses in packet headers led to the development of anonymizing proxy networks that hide the true destination IP address by first routing through proxies [5]. Current filtering technology is now employing *Deep Packet Inspection (DPI)*, in which characteristics besides the headers are examined, such as the contents. *DPI* techniques that can filter out specific Internet protocols. This has resulted in filtering-resistant services such as anonymizing proxies being specifically targeted to be blocked or throttled [13]. Traditional approaches to filtering resistance are no longer effective unless they also incorporate blocking resistance so that users can communicate with the circumvention services [6].

Traditionally, Internet traffic has been filtered using *shallow packet inspection*, where only packet headers are examined. Since packet headers must be examined anyway in order to route the packets, this form of filtering has minimal impact on the scalability of the filtering process, allowing for its widespread use. The primary means of categorizing packets for filtering with *shallow packet inspection* is to compare the source and destination IP addresses and ports to a known list. This can be either a "blacklist" of known IPs and ports that should be blocked, with the default being passing the traffic through, or it can be a "whitelist" of traffic that should be passed through with the default being to block. A blacklist is more commonly used in practice. The blacklists must be updated as new target IPs and ports are discovered. Port blacklists are circumvented using port randomization. In order to circumvent IP blacklists, anonymizing proxies are used that hide the true IPs by routing through a network of proxies that have IPs that are not on the blacklist. As the IPs of proxies are discovered by the adversary, they are added to the blacklist, so a fresh set of proxy IPs must be made available and communicated to users periodically. This is known as the *proxy discovery problem* and is an issue faced by all anonymizing proxy systems [7].

More recently, *DPI* techniques have been deployed that can successfully block or throttle most existing filtering circumvention solutions [26]. *DPI* filters packets by examining the packet payload and, while this is more expensive, it can achieve suitable scalability through random sampling of packets [3] or through only looking at the first packet or first few packets in a traffic flow. The primary test that *DPI* filters apply to packets is static string matching, although other fingerprints such as timing, packet length, and entropy are also possible. *DPI* can filter not only based on the content keywords, but also on the specific protocol, as determined by protocol fingerprinting techniques. Even encrypted protocols such as SSL/TLS can be fingerprinted. While encryption protects the contents from static string matching, encrypted protocols often include their own unencrypted handshakes preceding the start of encryption. Static string matching can then be used on the handshakes instead. For instance, SSL/TLS uses an unencrypted handshake for cipher negotiation and key exchange and so is easily fingerprinted and filtered. Current filtering hardware exploits this fact and can do SSL/TLS blocking using byte sequence matching. This capability was observed in the hardware study discussed in section 3.2.1 and Appendix E. Additionally, characteristics of the packets such as packet lengths and timing are not altered by encryption and can still be used for filtering.

1.4.1 Definitions

Filtering resistance is often discussed in connection with other related concepts such as anonymity, unlinkability, and unobservability. These terms are sometimes used interchangeably and sometimes given specific technical definitions. Pfitzmann proposed a standardized terminology that defines and relates these terms [25]. *Unlinkability* is defined as the indistinguishability of two objects within an anonymity set. *Anonymity* is defined as unlinkability between a given object and a known object of interest. *Unobservability* is defined as unlinkability of a given object and a randomly chosen object.

Defining properties such as anonymity and unobservability in terms of unlinkability opens the way for an information-theoretic approach. Hevia offers such an approach by defining levels of anonymity in terms of what information is leaked from the system to the adversary [16]. Unlinkability requires the least protection, hiding only the message content. Unobservability requires that no information is leaked whatsoever. Of particular interest is that an anonymous system of any type can be taken up to the next level of anonymity by adding one of two system design primitives: encryption and cover traffic.

1.4.2 Censorship-Resistant Publishing and Anonymizing Proxies

Circumvention technology has gone through multiple waves of development in response to changes in filtering technology. The first wave approach to achieving filtering resistance is through what is known in the literature as "censorship-resistant publishing". This work consisted of publishing services such as Publius [31], Tangler [30], and Mnemosyne [14]. A practical issue with the use of censorship-resistant publishing systems is that even a system that provides maximum protection for stored files must still be accessible in order for those files to be retrieved. If communication to the document servers is blocked, then the system cannot be used and is an ineffective means of circumvention.

Practical circumvention therefore requires protection for communications as well as documents. Protection of communications is the focus of the second wave of circumvention tools, which are known in the literature as *anonymizing networks*. The most well-researched anonymizing network is Tor [5]. The original goal of anonymizing networks was to provide anonymity for the communicating parties, but they have also found a use in circumventing filtering. Serjantov [28] proposed combining censorship-resistant publishing with anonymizing networks as a solution to attacks that block access to publishing servers. This solution compartmentalizes the problem by using the publishing system to protect documents and relying on the proxy system to provide resistance against communication with the publishing system being blocked. However, anonymizing proxies do not offer perfect resistance against all blocking attacks. While a network of proxy nodes can provide protection against destination IP blacklists, they are still vulnerable to various forms of *DPI* protocol fingerprinting.

Protection against protocol fingerprinting characterizes the third wave of circumvention research. This topic is dealt with by Kopsell, who proposes a method to extend existing anonymous publishing systems to bypass blocking, a property referred to as *blocking resistance* [21]. In light of the work of Serjantov and Kopsell it is evident that if anonymous proxies are a necessary component of censorship-resistant publishing and blocking resistance is a necessary property of anonymous proxies, then blocking resistance is necessary for censorship-resistant publishing.

Kopsel's threat model contains the assumptions that the adversary has control of only part of the Internet (the filtered zone), that some small amount of unblockable inbound information can enter the filtered zone (perhaps out of band), and that the blocking-resistant system design is known to the adversary.

Kopsell's solution is divided into two parts: access to the blocking-resistant system, and distributing information about the blocking-resistant system, which is essentially the proxy discovery problem again. The nodes in Kopsell's system are volunteer anonymizing proxies that clients communicate with over a steganographic protocol in order to obtain access to a censorship-resistant publishing system. Clients obtain an invitation to the network, including the IP addresses of some proxy nodes, through a low-bandwidth, unblockable channel into the filtered zone. Kopsell suggested using email to distribute the invitations. A number of ideas are proposed for the steganographic data channel such as SSL and SMTP protocols.

Though Kopsell's model for blocking resistance addresses some of the real issues facing censorship-resistant publication systems and anonymizing networks, it relies on the steganographic data and unblockable invitation channels to have certain properties that may not be met in actual implementations. The essential purpose of the steganographic channel is to provide resistance to protocol fingerprinting. However, even if the information cannot be recovered from the steganographic encoding, if it is discovered that the channel contains steganographically encoded information, then that channel can be blocked on this basis. In other words, the encoding must be undetectable in practice in order to be useful. While the goal of steganography is in fact to be in theory undetectable, Kopsell's approach essentially relies on the existence of adequate steganography in order to function and that the steganographic channel is not itself blocked for other reasons. The constraint on the invitation channel is that it is completely unblockable, as no particular protection is given to information distributed on this channel.

Analysis of historical attacks has shown that SSL is not a suitable encoding against modern adversaries inasmuch as the protocol is easily fingerprinted and blocked

[27][7]. Additionally, email is not a suitable channel for sending invitations because it is not blocking-resistant. Recent attacks have blocked the communication of IP addresses of proxies through email and instant messaging. Given these attacks on email and SSL, a potential research question for this approach is to determine what sort of channels are suitable for invitations and data to be communicated without being vulnerable to blocking.

Information theory provides a conceptual framework that offers an answer not just to the question of blocking resistance but also of its relationship to censorship resistance in general. Censorship-resistant publishing systems provide document unlinkability. Hevia connects the definition of unlinkability to information theory by defining it as the indistinguishability of information transmitted on the channels between the system and the adversary [17]. Boesgaard connects document unlinkability to information theory as the achievement of "perfect" secrecy, a technical term meaning that no information is revealed about the document through observation [2]. Censorship resistance can therefore be thought of as a form of perfect secrecy achieved by means of indistinguishability. Pfitzmann defines unobservability as a form of unlinkability [25] and Perng defines censorship resistance as unobservability [24]. In other words, censorship resistance is unobservability through unlinkability of the object of interest and a random object, which is equivalent in information theory to perfect secrecy. Viewed in this context, a censorship-resistant publishing system would be one in which the adversary cannot obtain sufficient information through observation of the system to distinguish which documents are accessed by users. Thus, censorship-resistant publishing is document unobservability. Anonymous proxies add a similar property, unobservability of the publishing system. The final step, which Kopsell calls *blocking resistance*, is unobservability of the anonymous proxy, which requires unobservability of the protocol by which clients communicate with the proxies. When these properties are combined, end-to-end unobservability is created from the client to the document.

The ideal communication protocol is therefore one that is unobservable, meaning that a packet or sequence of packets is indistinguishable from a randomly selected packet or random sequence of packets drawn from the set of hypothetical “normal” Internet traffic. This is not necessarily a steganographic encoding. A steganographic encoding is unobservable only so long as the message encoding is not detectable, regardless of whether the message can actually be decoded. Additionally, steganographic channels can be blocked if the cover channel is blocked. In the case of the rate limiting of Tor, SSL was being used as both encryption and as steganography since there is a large amount of non-Tor SSL traffic on the Internet. The Tor network has been successfully attacked in the past because of its use of SSL. One attack targeted unique characteristics of Tor's SSL handshake specifically and another involved a throttling of all SSL traffic [27][7]. While Tor has subsequently increased the steganographic strength of its SSL handshake by making it resemble Apache's SSL handshake, this does not prevent against an attack that blocks or throttles all SSL traffic.

Steganography is not the only option for unobservable protocols. Encryption is also a means of making messages indistinguishable. Specifically, one of the desired theoretic properties of encryption is that ciphertexts should be indistinguishable from each other. Although protocols such as SSL are encrypted, these protocols often have an unencrypted handshake. This unencrypted portion of the communication is primarily what is used to fingerprint and block the protocol. Additionally these protocols may leak other information to the adversary through other sources, such as packet lengths and timing. The set of indistinguishable objects is also limited, even in the best case. While ciphertexts are ideally indistinguishable from each other, and therefore theoretically so are encrypted protocols, they are distinguishable from unencrypted protocols. So an entropy measurement attack can be used to distinguish and block all encrypted protocols. A properly designed encrypted protocol with an encrypted handshake and unpredictable packet lengths and timings would be resistant to identification through fingerprinting, which the exception of entropy attacks targeting all encrypted protocols. In the normal use case for SSL, an entirely encrypted connection would not be possible as the communicating peers need to perform a public key exchange in order to determine the session key used to encrypt the conversation. However, unlike a normal SSL connection, Kopsell's model allows for a single out-of-band invitation to be sent prior to the establishment of the data connection. This affordance is the key to creating a secure protocol that is blocking-resistant even if the adversary monitors all other traffic and is aware of the design of the protocol.

1.4.3 Obfuscated Protocols

The goal of an obfuscated protocol is to provide protection from the adversary only so long as the adversary does not know the details of the obfuscation scheme. This is in contrast to a secure protocol that maintains protection even if the adversary knows the details of the protocol. What obfuscated protocols achieve is that new rules must be written to filter them and so protection is achieved until that occurs. Several obfuscated protocols have been developed with various goals, including blocking resistance. For instance, BitTorrent clients have implemented three encryption protocols in order to prevent filtering and throttling of the BitTorrent protocol. The most common of these in current usage is Message Stream Encryption (MSE) [32]. Although MSE has an encrypted handshake that facilitates the DH key exchange, analysis of packet sizes and the direction of packet flow have been shown to identify connections obfuscated with

MSE with 96% accuracy, primarily through analysis of the statistical properties of the key exchange [17]. As is evident from this and the following examples, the key exchange turns out to be the weak point in obfuscated protocols. As a typical key exchange is not encrypted, due to the fact that the encryption keys are by definition not available until the end of the exchange, the network traffic generated by the key exchange may be more vulnerable to protocol identification. Careful design with a specific focus on resisting protocol identification is necessary. This issue is explored further in section 4.3.2, “Encryption Layer Protocol.”

Obfuscated TCP (ObsTCP) has gone through several versions, each using a different means to communicate the keys, including TCP options, HTTP headers, and DNS records [33]. The strongest of these is DNS records inasmuch as TCP options and HTTP headers are readily blocked using static string matching, while DNS records are transmitted on a separate connection from the one carrying the data, requiring correlation between separate connections. However, an adversary has already demonstrated this sort of general correlation ability in the blocking of BitTorrent traffic by monitoring tracker protocol traffic to obtain the ports of the BitTorrent protocol connections and then subsequently interfering with the (sometimes MSE-encrypted) BitTorrent protocol connections [15][29]. A second connection from the same IP is not ideal as an out-of-band channel for the purpose of blocking resistance. Another protocol similar to ObsTCP called tcpcrypt does not have blocking resistance as a design goal and subsequently provides less protection than ObsTCP with a DNS key exchange as it uses static strings in the handshake [1].

An attempt has been made to address the handshake fingerprinting problem in the form of the obfuscated-openssh patch to OpenSSH that replaces the SSH handshake portion of an SSH protocol connection with a minimal blocking-resistant encrypted protocol [22]. An encrypted handshake for an existing encrypted protocol has the advantage that it involves the minimal amount of change necessary to achieve blocking resistance so long as the protocol already has resistance to other attacks such as packet size and timing. When circumventing filters that only look at the first packet or first few packets before classifying traffic, replacing the handshake may be sufficient protection regardless of what features the adversary is examining. This obfuscated-openssh handshake is designed to be resistant to fingerprinting by matching static strings and packet sizes. However, it is an obfuscated protocol only and not secure because its security relies on an assumption about the scalability of filtering technology. The handshake is encrypted with a key that is generated from a seed that is added to the beginning of the encrypted part of the handshake. The key is then generated by iterated hashes of the seed. The iteration number is chosen to be high enough that key generation is slow. The blocking resistance of this technique relies on key generation being too expensive to scale to all connections simultaneously. However, modern filters are capable of statistically sampling packets and processing them offline [3]. This approach is probabilistic in its ability to block connections, but is highly scalable. Additionally, the introduction of slow key generation may allow for less expensive timing attacks.

Between the beginning of this dissertation project and its conclusion, there have been new developments in obfuscated protocols. Much of this research has been motivated by the work of the Tor project to develop a *Pluggable Transports* framework to allow obfuscated protocol to be used as a wrapper around Tor traffic. The motivation is to facilitate academic researchers and practitioners in the field to develop a diverse variety of methods for obfuscated that can all be used to allow Tor to work on networks that block Tor traffic through protocol classification and filtering. The implications of *Pluggable Transports* for Dust will be explored in section 2.7, "Design Concept 5."

2. Design of Circumvention Tools Using Dust

The Dust engine is a general-purpose method for circumventing filters. The engine by itself is only useful to filtering circumvention researchers. To be used in an applied setting it must be incorporated into circumvention tools. In order to take Dust out of the laboratory and get it into the hands of users, a value sensitive design approach was used to build a customized circumvention technology for a specific community of users.

2.1 INTRODUCTION

Value sensitive design (VSD) is a theory and method for accounting for human values in a principled and systematic way during the design process [11]. It provides a means for making ethical considerations a part of the process of tool creation. This approach offers an alternative to the viewpoint of treating technology as politically and social neutral. VSD is a generalized approach to design that can be used to bring human values into the design of any technology. The VSD approach has been previously used in the analysis of design decisions in tool development for purposes of security and privacy. For instance, the Security Cards project [12] used a VSD approach together with heuristics from security threat analysis to foster a security mindset around the design and development of new technologies. The model for applying VSD to the creation of filtering circumvention tools based around the use of the Dust engine was a previous study done as a joint research project between the Information school at the University of Washington and the Computer Science department at Princeton. In this study VSD was used to improve the cookie management interface in the Mozilla Firefox web browser [10]. This is a relevant project in that it was a technological intervention designed to improve the security and privacy of Internet users. A similar approach was used here, customized to the needs of circumvention technology design.

2.2 Design Reflection Framework

Following the model for the Mozilla Firefox cookie management improvement project [10], the design process incorporated the following phases:

- Describe criteria for access to credible and relevant information
- In light of those criteria, summarize how filtered networks fall short in delivering such access
- Identify goals for circumvention of filtering to improve access to credible and relevant information
- Using each one of these goals, initiate an iterative design process:
    - Design new technical mechanisms
    - Implement new technical mechanisms
    - Formative evaluation
    - Repeat until the goals are met
- Reflect on the results of the design process and the strengths and weaknesses of using VSD instead of a conventional design process

Throughout the iterative design process, three types of investigations take place: conceptual, technical, and empirical. The conceptual investigations provide philosophically informed analyses of the issues central to the system under development. The technical investigations consider how existing technical designs provide affordances for values and how identification of specific values influences the design of new mechanisms that better support those values. The empirical investigations evaluate the system under design to provide a pragmatic grounding.

A key differentiator between the present research and the cookie management study is in how formative evaluation was conducted. Unlike the cookie management project, the purpose of building circumvention tools that use the Dust engine is not to evaluate the tools directly, but to build a realistic use case for evaluating the Dust engine and the filtering models. The fundamental difference is that the cookie management study was evaluating the usability of an interface, while the present research is evaluating the effectiveness of a circumvention tool in realistic scenarios. Therefore the formative evaluation was to use as the metric of evaluation the efficiency and effectiveness metrics developed in Section 2.2.2 on evaluating the engine. The third stage of research on building circumvention tools built on the previous evaluation involves developing a working circumvention tool with properties derived by the VSD process and then evaluating efficiency and effectiveness in this context. The result of this work was an engine and set of models that describe a realistic filtering and circumvention scenario, as well as a functioning tool ready to be deployed to users. Before building a technology and evaluating it through technical metrics, several iterations of design reflection were undertaken. The landscape of filtering and circumvention technology is always changing. As new information becomes available about deployed filtering techniques, and as interaction with users elucidates the design goals, new iterations of the design concept were made. This chapter considered the initial design reflections leading up to the final design. Evaluation of the final design is presented in the evaluation chapter.

2.3 Design Concept 1

Motivation

The original idea for a circumvention tool was conceived during the "Arab spring" revolutions of 2011. At this time, American media gave credit for the revolutions to Twitter for enabling both coordination of protests and dissemination of news and statements from the protesters to the world at large. Due to national media blackouts and restriction of access for international journalists, information regarding the revolutions through traditional news outlets was limited and Twitter became a primary source of news for Americans seeking to follow the folding events. It was possible to imagine by extension that Twitter was a primary source of credible and relevant information for people residing in the Arab Spring countries. Unfortunately, Twitter was subsequently filtered in many of these countries.

Design Concept

The design concept was a proxy using Dust that would allow access to Twitter from Arab Spring countries. Through Twitter, credible and relevant information about what was happening in those countries could be accessed.

Embodied Values

The core value of allowing access to credible and relevant information was present in this early prototype, although the scope was greater than just a proxy. Also included in the project was an archive of all information related to the Arab Spring revolutions. This version of the project was also focused on posting of information as reading it.

Implications for the Dust Engine

An important consideration at this time was that the size of a tweet is very small, 140 characters. This can easily fit into a single UDP packet. The first version of Dust therefore concentrated on conversations consisting of a single UDP packet. UDP transport was implemented and TCP transport was not. Also at this time Dust was compatible only with IPv6; IP6 over IPv4 tunneling, such as Teredo, was necessary to use it on IPv4 networks.

Design Reflections

One issue with the values embodied by this solution was that some participants in the Arab spring revolutions considered Twitter to be symbolic not of freedom of communication, but of American cultural imperialism. They claimed that the conversations on Twitter did not represent what was going on in their countries, but rather that of expatriates living in America, which they argued were not representative of local sentiment. They also claimed that Twitter provided no organizational role in the protests. A second issue is that Twitter is a centralized, commercial service that has no particular interest in use as part of organized poltical revolution. It is therefore not designed with this use case in mind and lacks some desirable features.

Of primary importance was a technical issue with having users log into Twitter using a proxy service. There are two ways to authenticate with Twitter. The first is to furnish the username and password for the account to the application. This method has poor security properties and is no longer used by most applications. The method Twitter prefers developers to use is OAuth, a protocol by which a user can authorize an application to have access to their account by means of an access token. The app presents the access token to gain access to the account, but the user can revoke the token at any time. This method has better security properties. However, the way OAuth is implemented with Twitter, authenticating an application requires that the user visit the Twitter website. Since the Twitter website is filtered in this use case, in order to create a Dust-enabled Twitter client a Dust-enabled web browser must also be created. This is prohibitively complicated for the use case. A second version of this application was also attempted using open source software for running a service similar to Twitter called StatusNet (now renamed to pump.io). This software is similar to Twitter, but is decentralized in the sense that anyone can install their own StatusNet service.

Unfortunately, StatusNet also uses OAuth for authentication and so has similar problems. However, since it is open source it can be modified to allow for a more convenient authentication mode. The question then becomes whether access to a Twitter-like service is what is important or if only access to Twitter itself has value. A more fundamental question is whether or not people in Arab Spring countries do in fact use Twitter as a primary source of credible and relevant information.

2.4 Design Concept 2

Motivation

The goal was to replace the centralized Twitter infrastructure with a decentralized publishing medium. Also important was providing a means of authentication for authors that could work on a filtered network.

Design Concept

A blog publishing setup was created that allowed posting using messages encoded using Dust. Authentication for these messages used public key cryptography and digital signatures, so no separate communication with a web server was necessary to authenticate the author. When a new message arrived for posting, the attached digital signature could be checked to verify the author before the post was made public. The Octopress blog management software was used because, unlike other popular blogging solutions, its output is static HTML files that can be hosted anywhere and mirrored easily in case the original hosting site is blocked or taken down.

Embodied Values

This design concept valued decentralization over centralized services. Therefore, the emphasis was on the technical mechanism of distributing information rather than on the social network provided by services such as Twitter. It also values protecting posting over protecting reading. Protection of reading was facilitated by making it easy for third parties to set up mirrors of content. This assumes a widespread interest in the content and a network of volunteers to set up mirrors and distribute their addresses to readers. Here the Internet is being treated much as Twitter was portrayed in the Arab Spring, as an alternative to print and television news. While hosting of content is more decentralized in this design, production of content is in some sense more centralized. While anyone is free to run their own blog site, the necessity of finding and keeping up with blogs makes it more difficult for every person to have their own blog, as compared to the ease of use of a Twitter account.

Implications for the Dust Engine

This design focuses entirely on publishing and not reading. Circumvention of filtering on the reading side is provided by mirrors of the sites. Dust is only responsible for transporting the posts when they are originally posted. This changes the traffic profile considerably. Compared to tweets, posts are significantly longer. 140 characters would be considered quite short according to blog post standards. A single UDP packet is therefore no longer a viable means of transport for posts. Therefore, both TCP transports and a layered protocol that allows longer messages to be sent over multiple UDP packets were implemented.

Design Reflections

A major change in this design was a focus on publishing rather than an equal emphasis on publishing and reading. An important question is whether a new publishing platform would be adopted by users. Users could possibly prefer to publish on Twitter because it is more convenient for promoting posts. Since Twitter publishing has technical challenges in this use case, an alternative publishing platform was developed that overcomes these challenges. However, getting users to migrate to a new platform may prove to be an even more difficult challenge.

2.5 Design Concept 3

Motivation

The third iteration came from a discussion with a user on a filtered network who posted on a circumvention tool mailing list about where people in her country get their news. The American news media promoted the idea that people in Arab Spring got their news about current events from Twitter, but some people from Arab Spring countries said that this was not so. In this particular conversation, the poster said that people in her country get their news mostly from blogs. The major news sites are state controlled, but there are some independent news blogs. They are hosted in the United States because they would be shut down if they were hosted in the country with the filtered network.

Design Concept

This iteration was therefore a news reader. Creating a news reader instead of a Twitter proxy solved some of the major problems of previous iterations. News sites are public and so do not require authentication. While news sites can be read using a web browser, many also support the RSS protocol for downloading news articles to read in a dedicated news-reader application. In fact, many news sites support RSS. Supporting only RSS is much simpler to do securely than supporting all of the operations available in a full web browser. An RSS processor was created that could download RSS files from web servers, sanitize them to remove any possible security problems, and package them in a compact format for transmission to the user.

Embodied Values

This design iteration flips the emphasis from publishing to reading. There is no capability for publishing in most RSS news readers, so only reading is supported over Dust and publishing must be accomplished through other tools. This is a major change from the last version, which supported only publishing over Dust. The reason for this change is the realization that the problem facing news sites for countries with filtered networks is not publishing, but reading. By hosting the sites outside of the country with the filtered network, they are relatively safe from being taken down. Only a small number of people need to be able to publish to the major news sites and they can even do so from outside of the country. The readers are the majority of filtered users and so reading is the primary activity to protect in order for news sites to be a practical source of credible and relevant information for people in the country.

This design is also a reversal of the previous design goal of seeking decentralized platforms. While news sites are more decentralized than Twitter, they are less decentralized than individual blogs inasmuch as there are only a few major news sites. This is a compromise in terms of centralization and decentralization that is intended to represent a good balance point between flexibility and usability. Fortunately, the primary reason for seeking decentralization in previous designs was to allow for a means of decentralization authentication that could work practically on filtered networks. Since this iteration supports only reading and not publishing, no authentication is necessary. Therefore, the primary problem for this project caused by centralized platforms has been solved by loosening the requirements.

Implications for the Dust Engine

RSS traffic has a similar profile to loading a website or downloading a file. RSS is carried over HTTP and the content is XML with some embedded HTML and hyperlinked images. The traffic is therefore similar to normal HTTP traffic, making HTTP a good profile to emulate on networks that allow general HTTP but specifically block RSS or specific news websites. Fortunately, HTTP is one of the most studied protocols and much data exists from which to build models. On networks that filter HTTP generally, another profile with similar characteristics must be chosen instead. Mail protocols such as POP and IMAP, for instance, have similar profiles in some ways to HTTP.

Design Reflections

The difficulty for this version of the project came when considering the user interface for the news reader. There are a variety of RSS-compatible news readers in existence and each has a very different user interface with features optimized for different uses of the RSS technology. For instance, some users subscribe to hundreds of different news sites, many of them blogs about specific niche topics. Others subscribe to a few major news sites. Some users use RSS news readers primarily to read webcomics and do not use them for news sites. Some users read every article, while others skim titles and only read a few articles. Users of RSS news readers had a diverse variety of responses when asked about what features are essential to an RSS news reader.

2.6 DESIGN CONCEPT 4

Motivation

The next iteration was motivated by talking to circumvention tool users and developers at a private conference gathering circumvention technology researchers to discuss the landscape of the field. The refrain heard over and over again from tool developers is that users do not want to use new applications. They want to enable the users interfaces they already use to work on filtered networks.

Design Concept

Instead of developing a unique user interface for the news reader application, in this iteration it has been converted to a proxy that enables an existing news reader application to work over Dust.

Embodied Values

Though this application is implemented as a network proxy, it is not designed to be a general-purpose proxy for all network traffic. It is an application-specific network proxy that is designed to only carry specific traffic.

Implications for the Dust Engine

For most of the Dust engine, it does not matter if there is a custom interface or not. The one layer that does change is the application protocol layer. In previous iterations, a compact message-based protocol was used, tailored to each specific application. However, in order to act as proxy a stream-based protocol is necessary. A general-purpose stream-based protocol was therefore developed.

Design Reflections

This design on the one hand simplifies the application in that it no longer requires a complex user interface, and on the other hand complicates the application as it must now support a more generalized streaming protocol instead of a simple application-specific message-based protocol. This may be overall for the best as this added complexity is entirely hidden from the user.

2.7 Design Concept 5

Motivation

A non-profit organization that studies Internet freedom issues released a report on the state of Internet freedom in one country with a filtered network [34]. It covered not just the filtering in place on the network, but also a broader strategy of control. In addition to filtering, more active attacks are also being used. A national web browser and an operating system are being promoted to users. These tools undermine the basic privacy and security properties provided by HTTPS when used normally, allowing for the filters to decrypt encrypted traffic.

This attack is known as Man-in-the-Middle (MitM) and is one variant of a family of attacks that allow for decryption of intercepted encrypted messages. In this variant, the national web browser is configured to accept certificates from the national Certificate Authority (CA). The CA then generates new certificates for all websites. One way to think of these is as counterfeit certificates. They are generated by the national CA without any relationship with the real owner of the website for which the certificate is being generated. The goal of the national CA in generating these new certificates is that, unlike the original certificate, the national CA knows the private key for the counterfeit version. The CA shares the certificate with the national browser and with the filters. The private key for the certificate is also shared with the filters. The filter administrators combine their standard filtering hardware with a specialized device known as an SSL tap. The SSL tap stores the counterfeit certificates and corresponding private keys. Whenever the filter encounters an HTTPS connection (or anything else encrypted with SSL), it sends the traffic first through the SSL tap. The SSL tap does not pass the traffic through. Instead, it first pretends to be the client. It negotiates an SSL connection with the server, obtaining the decrypted content just as a web browser would. It then returns this decrypted content to the filter for filtering. When the filter is done examining the traffic, and perhaps filtering it, it sends the modified traffic back into the SSL tap for a second pass. In this second pass, the decrypted traffic is re-encrypted, this time using the counterfeit certificate. The SSL tap then pretends to be the server and negotiates an SSL connection with the client. The client obtains encrypted traffic and decrypts it normally. The only difference from the client's perspective is that the certificate has changed. The efficacy of this attack depends on how the client deals with this situation. If the client does not accept the national CA as a valid authority, then it does not accept the certificate and is not able to make a connection to the server. This is why a national web browser and a national operating system that distributes this web browser is necessary to ensure that the web browser accepts certificates generated by the national CA. If the client accepts the certificate, then it allows the connection. What the adversary has achieved is the ability to examine all encrypted traffic as if it were plaintext, without any warning to the user.

Design Concept

The basic design concept remains the same, a proxy that allows downloading of RSS news feeds for reading in a news-reader application. A question remaining from the previous iteration of the design concept was what specific RSS news reader should be used. In this iteration, desktop news readers were eliminated from consideration in favor of a mobile news reader application for the Android platform. The desktop was previously an attractive platform to target for this tool due to the widespread market penetration of desktop computers. However, with the advent of the national operating system, deploying circumvention tools to desktops could be more difficult. A mobile operating system was therefore chosen as an alternative as the country with the national operating systems for desktops has not yet developed a national mobile operating system.

The possible difficulty in making this transition is that mobile news readers have a more limited feature set than their desktop equivalents. One of the missing features on some mobile applications is the ability to configure the application to use a proxy. Fortunately, one news reader application was found that supported configuration of a proxy, Courier by the Guardian Project. It has been developed specifically for use with the Tor anonymizing proxy and therefore implements proxy support. The choice was therefore made to use Courier as the news reader application and to develop a compatible proxy that enables Courier to be used with Dust traffic encoding.

Embodied Values

This iteration of the design aligns the tool more closely with the community of practice around circumvention tools. While the choice of Courier was a pragmatic one as it provides the needed feature of compatibility with proxies, it is also an application created by the Guardian Project. Guardian is an active member of the circumvention tool developer community and works closely with the Tor project. Due to this close integration between Guardian applications and Tor, the simplest path for integrating Operator with Courier also allows for easy integration with Tor. While integration with Tor is not a direct goal of this design project, Tor is leading in research and development of obfuscating protocols with the development of the *Pluggable Transport* framework for adding network protocol obfuscation to existing applications. Compatibility with Tor therefore brings Dust in line with the emerging standard for circumvention tool development. There is a tradeoff here in terms of embodied values. In the original conception for Dust, efficient purpose-specific application layer protocols were used. The intention was to use transports that were tolerant to extreme networking conditions such as high latency, low bandwidth, and unidirectional traffic flow. These conditions are opposite of what is required for Tor, which is a low-latency, high-bandwidth, bidirectional traffic flow protocol. Dust was also originally message-based, whereas Tor is connection-based. In adopting a proxy approach and requiring compatibility with existing software, the technical requirements for the transports used by Dust have changed, as has the application layer protocol. Dust is therefore now compatible with Tor and can ostensibly be used to proxy Tor connections. Dust has therefore moved from being an approach that was incompatible with existing tools, to being compatible with the major tools currently in use. While there may be a tradeoff in efficiency and quality of service during extreme network conditions, it is now easier to compare Dust to existing obfuscating protocols already used with Tor.

Implications for the Dust Engine

In some countries with filtered networks, HTTPS is no longer always going to work as a protocol that can bypass filters. Dust can shape traffic to resemble any protocol, including HTTPS. However, this resemblance is only on a statistical level. While this works for filters, it does not work for SSL taps. The SSL taps interactively negotiate SSL connections with servers. Therefore, real SSL is required rather than just a statistical model. The SSL is then stripped away and replaced with a new SSL wrapper, destroying any information Dust might have encoded in the SSL layer. To use HTTPS, Dust would need to encode to an HTTP shaping target and then wrap this in SSL. While this could be done, it would provide no advantage over just using plain HTTP. Therefore, plain HTTP and other unencrypted protocols that do not use SSL have gained some status as preferred targets for Dust encodings.

In terms of implementation, the most convenient way to deploy Dust to work with Courier is to package them together. Android applications must be deployed in the form of packages and it is difficult to have one package depend on another. The common practice on the Android platform is to distribute self-contained packages. The Guardian Project has already done the hard work on packaging Courier so that it can detect and use a proxy if a proxy is installed. Specifically, Guardian distributes an Orbot package that provides a Tor proxy. Courier and other Guardian apps detect if Orbot is installed. If it is installed, these apps automatically use the Tor proxy bundled with Orbot. This by itself is not helpful in getting a Dust proxy integrated with Courier. However, current versions of Orbot also ship with obfs4proxy, a bundle application that implements the *Pluggable Transports* specification. It includes several obfuscating protocol implementations that wrap the Tor connections, such as obfs3 and obfs4. Since Orbot and obfs4proxy are both open source, the simplest integration path is to include Dust into a new version of obfs4proxy and include this version in a custom Orbot package. When installed alongside Courier, Dust can be used to fetch RSS news feeds. The obfs4proxy application conveniently does the hard work of implementing the Pluggable Transport specification. It provides a simple API that Dust can use to provide protocol obfuscation services. This is a major change for the architecture of the Dust implementation as Dust is now included in obfs4proxy rather than being integrated directly into each application. The application now needs to support using either obfs4proxy directly or Orbot rather than using Dust directly. Conveniently, several applications already exist which do this, mostly maintained by the Guardian Project.

Design Reflections

One effect of these revelations on the Internet control strategy for some countries with filtered networks is to confirm that the current direction is the right choice in several ways. RSS news readers are one way to read news, but direct access to the news websites through a web browser is also an option. In light of the adoption of a national web browser in at least one country with a filtered network, the latter approach does not seem to be preferable. The forefront of circumvention tools may be in mobile apps, at least for the moment.

2.8 DESIGN CONCEPT 6

Motivation

Deciding to implement Dust as a *Pluggable Transport* brought scoping constraints that simplified the design decisions for the project. The remaining decision to be made was on what models to use with Dust to create a Pluggable Transport effective for transporting news-reader traffic across the filters. One of the lessons learned from the process of implementing and testing Dust as a Pluggable Transport was that there is a downside of shaping packet timing. In order to obfuscate the properties of the tunneled traffic, Dust always follows the model rather than the tunneled traffic. This includes sending packets when the model specifies it should send packets rather than when the tunneled traffic actually has data to send. Dust traffic is therefore at a constant bitrate (with random variation). This traffic profile is very different from news-reader traffic, which follows a request/response format. From a bandwidth perspective, this looks like a burst of high bandwidth in request direction (and silence in the response direction), followed by a sustained high bandwidth response (and silence in the request direction). This turns out to be a worst-case scenario for the way Dust does shaping of packet timing. It averages these bursts into a constant bitrate held for the duration of the connection. This means that the time it takes to send the request is the normal duration of a whole connection. The request is sent too slowly, and since the response blocks on the completion of the request, this makes the overall connection slow. In order to achieve enough effective bandwidth, the transport must therefore be configured to use a higher average bandwidth than the application. In the case of news-reader traffic, the request side must be set high enough that the request can finish. Since the Dust transport consumes constant bandwidth, this amount of bandwidth is in constant use regardless, even after the request has completed. Fortunately, a news reader is a low-bandwidth application, so the constant bandwidth use is fairly low. However, complications are caused by the fact that the Courier news reader uses Tor and that in this use case Dust wraps Tor protocol rather than the news-reader traffic directly. The Tor protocol requires a minimum bandwidth and latency in order to work correctly, regardless of what application it is transporting. Therefore, the constant bandwidth had to be set to something appropriate for Tor to function rather than just enough for RSS.

Design Concept

Packet timing shaping was dropped from the design concept in this iteration inasmuch as it was too expensive to be practical. Models using the remaining features of byte sequences, content distribution, entropy, and packet length were then developed for the use case of the Courier news reader for Android. Three different models were chosen: HTTP, HTTPS, and RTSP. HTTP is an obvious choice because, historically, there was an incident where all protocols except for HTTP were blocked. Therefore at least one HTTP model is a necessity. HTTPS was chosen because it is the most popular protocol right now for mimicry. Existing tools such as Tor mimic HTTPS by encapsulating the application protocol inside of SSL, similar to how HTTPS is HTTP encapsulated in SSL. RTSP was chosen to show that it is possible to use Dust with a variety of protocols, not just the standard choices of HTTP and HTTPS. RTSP is a practical choice for this use case because it resembles what Tor traffic already looks like more closely than HTTP or HTTPS.

Embodied Values

In may seem that due to the choice to make Dust into a Pluggable Transport, there is a sense in which the specific circumvention tool is no longer as important. The Dust Pluggable Transport is a generic encapsulation for Tor traffic, which is itself a generic encapsulation of traffic for a variety of applications from web browsers to news readers. However, the Dust engine is only one part of the overall Dust system. Just as important as the engine is the set of models. The efficiency of the models depends on the match between the model and the application protocol being encapsulated. Therefore, while the Dust engine is generic, the models are not. These models were chosen for the specific use case for news-reader traffic carried over Tor. Other applications with a similar traffic pattern may work equally well. However, applications with very different characteristics most likely have poor performance, perhaps to the point of being non-functional. Therefore, there are still choices to make that carry embodied values. The choice to implement the Pluggable Transport standard just moved these value-laden choices from the software into the models.

The choice of transports that were provided also has embodied values. HTTP was chosen to plan for the worst-case scenario: everything is blocked except for one protocol. It is the fallback choice. HTTPS was chosen because it is the popular choice. It is likely to work in common scenarios other than the worst-case scenario. It is also easy to compare Dust using HTTPS to other obfuscation methods since they also often mimic HTTPS. It is the practical, mainstream choice. RTSP was chosen because it has better characteristics for matching Tor traffic than HTTP or HTTPS. It was also chosen because it is a not a popular choice for obfuscating protocols. It represents an alternative choice that is experimental. By providing multiple models, several different, perhaps conflicting, values can be embodied simultaneously and users can choose those that are appropriate for their situations.

Implications for the Dust Engine

This iteration had minimal effect on the Dust engine as it was mostly a matter of choosing models, which are interchangeable within the engine. The one significant change to the engine was to stop doing shaping of timing. This is a minor change on paper, but in terms of implementation was a major architectural change because it affected the scheduling semantics of packets. Fortunately, this was one of the more difficult architectural differences to implement between Dust and other transports, so eliminating it simplified the architecture. Most transports use a “push” architecture in which new transport packets are sent only when new application data arrives. Since Dust sent transport packets according to the model rather than based on what the application level traffic was doing, it had to have entirely different scheduling semantics based on timers. Eliminating timing shaping allows Dust to use the more traditional push architecture, making it more similar to other transports. Therefore the existing *Pluggable Transport* code is a better fit and the overall Dust code is simplified.

Design Reflections

This concludes the major design decisions for the project. The engine has been stabilized and the models have been chosen. Whether these models turned out to be good choices is explored in the evaluation section.

2.9 CONCLUSION

The design reflection process occurred throughout the duration of the project. Here it is presented as a linear process as if all design reflection occurred first, followed by implementation and then evaluation. In actual execution, these processes happened simultaneously. All of the designs were to some extent implemented and evaluated. The design reflections therefore represent an immense amount of engineering effort. A typical research effort would most likely have stopped after the first prototype was built, as this would be sufficient for a research paper. However, the value-sensitive design approach requires a higher standard of justification and as a result six different design concepts were implemented. The final result might seem somewhat mundane as it is essentially a *Pluggable Transport* for Tor. Several research projects started with the goal of designing a *Pluggable Transport* for Tor from the outset and so achieved this with a much smaller expenditure of time and effort. However, the value of design reflection is not just what ends up being built, but also learning from all of the design concepts that were in the end found inadequate. The overall takeaway from this process is that the circumvention tool space is conservative about new tools. The most viable approach is to make the minimum intervention necessary to enable an existing tool to work on filtered networks. Now that *Pluggable Transports* has emerged as a standard, this is the obvious target for future work. There are trade-offs here, and designing for the *Pluggable Transport* standard brings some constraints that may eventually turn out to be undesirable. However, in the near-term, this is probably the best approach as the key to adoption is to have your transport deployed as even the best possible transport is not useful if no one is using it.

It was a long process to implement six design concepts. It is of interest to see how the embodied values changed throughout the process. The goal from the outset was to provide access to credible and relevant information. Existing networks fail to achieve this goal because of filtering. The first design concept was a proxy to access Twitter. However, discussion with users revealed that Twitter was not the primary source for credible and relevant information, but rather blogs were used. The next concept was a way to publish blogs. However, further discussion with users revealed that filtering of publishing was not the major issue, but rather filtering of reading. The next concept was to create a news reader, but discussion with users showed that they did not want a new application but rather for their existing applications to work on filtered networks. The next concept was to create a proxy that enabled a desktop news reader to work on filtered networks, but a new report revealed that the launch of the national operating system and national web browser in one country with filtered networks destroyed the security of desktop apps. The next concept was to deploy a proxy for a news reader on a mobile operating system, avoiding the issues with desktop application security. This was the final design for the engine, at which point the next consideration was what models to use. In the final design concept, one feature (packet timing) was eliminated in order to approve efficiency and then three models were chosen: HTTP, HTTPS, and RTSP.

Overall, the core value besides the original goal of access to credible and relevant information has been user-centered design. Whenever the problem with a design was not technical in nature, it was an issue of user adoption. Three questions could be asked that guided the whole design process:

- How do the users access credible and relevant information?

- What platforms can be used for deploying tools that do not undermine credibility?
- What tools will users actually use?

These are three questions that can be used in future value-sensitive design processes for the development of other filtering circumvention tools. So this represents a general learning from incorporating this design process in addition to the specific design concept that was developed as the project for implementation and evaluation.

3. Model Building

3.1 Introduction

The core of Dust is to develop models of how filters work such that circumvention tools can be automatically generated to circumvent those filters. This requires a combination of semantic and statistical modeling. On the semantic side, modeling requires determining which features are significant. On the statistical side, observations of each feature are generalized into a probability distribution for that feature. At the highest level, a model of a filter consists of two categories: blocked traffic and allowed traffic. These categories can be subdivided into a hierarchy of subcategories, down to the level of individual protocols. The specific taxonomy is unique to each filtering device.

Drilling down into the taxonomy of protocols, the filtering rules for each protocol can also be modeled. The first decision when modeling a protocol is to determine which observable characteristics of the protocol are significant. Different protocol classification techniques use different observable properties. Depending on the filtering device and the specific protocol, different properties are significant. Once a set of properties has been chosen, a probability model can be constructed for each property. The use of a probability model allows for individual observations to be generalized. Observed data is used to generate the probability model and then the circumvention tool uses the probability model to generate new traffic. By using probability models drawn from the filter model's set of allowed traffic, the traffic that is generated has characteristics resembling allowed traffic across all modeled properties for that filter. As the generated traffic is categorized by the filter as allowed traffic, the filter allows it to pass through.

An important difference between the models used here and those used in other unobservable communication research is that what is being modeled is not solely the characteristics of each protocol, but also the significant characteristics as defined by the filter. Properties that the filter does not use for classification purposes are not included in the model. The probability models of protocols are the models of the protocols as seen and understood by the filter. In this sense they are actually models of different aspects of the filter and not of the protocols themselves. In contrast, other work in the field focuses on modeling protocols regardless of the context of the specific filter. As an example of the difference, with this approach if a filter blocks some HTTP traffic and allows some HTTP traffic, then these would be represented as two different protocols ("blocked HTTP" and "allowed HTTP") with different probability distributions. The conventional approach would be to model one protocol ("HTTP") with an attempt to find a protocol that is always going to be classified into an allowed category. Unfortunately, that approach is not viable given how filtering works. There is no single protocol that is guaranteed to be safe. Even traffic using an individual protocol such as HTTP can be further subdivided across any of its significant properties to selectively filter only some traffic using that protocol. An adaptive approach in which the protocol models can change as easily as the filters can be reconfigured is therefore more versatile.

3.1.1 Goals

Modeling is a process of abstraction and estimation. By the nature of this process, there are discrepancies between the model and the modeled object. Evaluation of a model therefore requires a context that determines which features are important. Within this context, models can be compared in terms of goodness of fit according to a chosen metric across the chosen set of features. However, comparable models are not always available. In the case of this research, there are no previous studies that have modeled the same adversaries with the same metrics to compare against. How then can the score given a specific model be interpreted to determine if the model is a good model or a poor one?

The soundness of the adversary models is first argued by way of the means of construction. It was shown that the features chosen for the models are based on empirical research into filtering hardware. The specific modeling for each feature was based on an empirical field study where data was collected in 4 countries. The field study is discussed in depth in section 3.2.2 and the results are shown in Figures 1 through 18 and Illustrations 1 through 4. This data was used to build a test model using the chosen feature set. The field study data was used to determine the underlying distributions for each feature that was used to fit the models. Then, a model was constructed using those distributions to test overall goodness-of-fit. This model distinguishes between two types of traffic: HTTP and HTTPS. The scores for this model were used as a baseline for understanding scores for adversary models.

When scores for model goodness-of-fit were interpreted, two methods were used. First, the goodness-of-fit for individual features was examined. These scores can be compared across models but not across features. This method was used to compare different adversary models on a feature-by-feature basis. A difference in score between two models does not necessary mean that one is better than the other. Different adversaries focus on different protocols and for each set of protocols the features may carry varying amounts of information. For instance, entropy is primarily useful for distinguishing between encrypted and unencrypted messages. Therefore it is a useful metric for distinguishing between HTTP and HTTPS. However, if an adversary were attempting to distinguish between two encrypted protocols, then the entropy feature carries less information in that context. The adversary using entropy to distinguish between two encrypted protocols would perform more poorly not due to the construction of the model, but due to the decreased available information. Therefore, where feature scores are useful is in comparing specific features across different models to determine the efficacy of different features for distinguishing sets of protocols. A wide variance in the goodness-of-fit for a specific feature might indicate that a more optimal distribution function might exist. For instance, a normal distribution was originally used for modeling durations, but it was determined through fitting against the empirical data that an exponential function is a better fit.

The second metric of evaluation was the overall accuracy of the model in correctly classifying traffic. This was tested in the models by means of two-fold cross-validation, in which connections are assigned randomly to two equal-sized training and validation groups. The training set is used to train the model and then the model is used to predict the data in the validation group. The testing and training groups are then switched so as to build and validate a second model. The cross-validation process results in two different models for each feature of each protocol. Later, in the evaluation stage, these two models were each used to build simulated adversaries and protocol obfuscating encodings. Each encoding of the two encodings can then be tested against each of the two adversaries, resulting in four test results that measure the effectiveness of the encoding.

In summary, the goal of model building was to demonstrate that the models have been constructed in a sound manner based on empirical evidence for the feature set and the distribution functions for those features. Additionally, the goodness-of-fit and accuracy metrics were used as baseline measurements. Cross-validation was used to show that the models accurately represent the observed features without overfitting.

3.2 Preliminary Research

Before commencing the statistical analysis and goodness-of-fit tests, the semantic aspects of the models needed to be determined and data needed to be collected on which to do statistical analysis. The semantic analysis was done using a laboratory study of two hardware filtering devices. The results of this study determined what features of network traffic should be collected for statistical analysis. The data collection was done using a field study in 4 countries, focusing specifically on the chosen features. Once was the preliminary research was done, the collected data was used to for building models to be tested in the evaluation section.

3.2.1 Hardware Study

Preliminary research on model building was done by investigation of filtering hardware. Two filtering devices were studied: a Blue Coat PS12000 and a Procera PL8720. Blue Coat is a popular vendor with wide deployment. Procera is a smaller vendor with more advanced devices. Studying these two devices provided for a practical and grounded look at the state of filtering technology. The devices were first studied by reading the product documentation and asking questions of the support engineers.

Once the basic capabilities of the devices were determined, a set of utilities to generate test traffic was written. These tools are open source and available for researchers to use. They are released under the name “Dust-tools”. Information on downloading the software written in the course of this dissertation, including Dust-tools, is available in Appendix D. Details of how the Dust-tools utilities work and how to use them can be found in Appendix E.

Observations of how the test traffic was classified by the devices allowed a semantic model to be built. This model captured both the taxonomy used for classification and the set of rules used to classify traffic into those categories. This research was the basis for the semantic component of the filter models. The significant characteristics of traffic used by these devices for protocol classification was used as the template for building filter models.

The results of the hardware study show that filtering technology in the field is not as advanced as the hypothetical adversaries discussed in the unobservable communications literature. The primary methods of classification are based on shallow examination of the packet headers with the primary use of Deep Packet Inspection being matching of static strings on a per-packet basis. Entropy and packet timing are used, but in a more limited way. Use of packet length for classification was not observed in these devices. Additionally, only the first few packets of a flow are considered. Once a flow is classified, further packets are not examined. There are additional constraints on the abilities of classifiers. For instance, both sides of the connection need to be visible in order for classification to work.

In order for filtering hardware to be effective at classifying real-world traffic, it requires customization by the user. A combination of connection properties can be assembled to create a traffic profile that can identify targeted traffic, based on automatically identified service information, statistical properties, and packet header information such as IPs and ports. For the Blue Coat device, no statistical information is available for custom filters. The automatic service discovery is therefore of primary importance in creating custom filters. For the Procera device, several statistical properties based on packet size and flow direction are packaged into a few set categories such as "Download", "Unidirectional", and "Streaming". These connection flags can be used to create custom filters that, for instance, block unclassified protocols that are "Random looking". A key element of creating custom filters is that unclassified traffic is put into its own category (either "Unknown" or "default", depending on the vendor) and this category can be filtered in the same way as an identified protocol. All unclassified traffic can be filtered the same way, or custom filters can break apart the unclassified traffic based on IP and port information, using either whitelists or blacklists for filtering.

The customizability of these devices means that there is not one particular protocol that can circumvent every device. Each device is configured to block certain protocols and to allow certain protocols to pass. An approach that mimics an existing protocol such as SSL, even if it is entirely effective at mimicry, is an ineffective encoding if all SSL is filtered. Similarly, encodings that do not match any recognized protocol are ineffective if all unclassified traffic is filtered. However, whether unclassified traffic or SSL are filtered is dependent on the network inasmuch as each device is configured individually.

The definitive way to know what works on a given network is to try different encodings and see which ones get past the filter. Once the set of blocked and passed protocols has been discovered, Dust can be configured to encode traffic to either mimic a set of protocols that are allowed or, if unclassified traffic is passed, encode traffic to avoid sharing characteristics of the set of protocols that are blocked. Even when empirical data is not available, the filtering devices have specific constraints in their functionality that ensure that once the protocol partition has been determined a suitable encoding can be generated. In the case of the Blue Coat device, all that is necessary is to either produce or avoid a static string in the right offset of one of the initial packets. If assigning the flow to the unclassified category is sufficient, then simple content encryption, which avoids any static strings whatsoever, should be effective. For the Procera device, the signatures were more complicated. In addition to dealing with the same static string issues as the Blue Coat device, it could also be important to shape the traffic to enable and disable the correct set of flags by changing the statistical properties of the content, size, and flow direction. Shaping these properties to achieve the correct set of flags could be necessary even if the protocol was unclassified as the custom filtering rules can further category unclassified traffic based on flags.

Finally, it is of particular importance to note that protocol obfuscation alone is not sufficient to bypass filters if servers with known IPs are used. Classifying traffic based on IP is a core piece of functionality of filtering hardware and new *DPI* approaches do not make this technique obsolete, but rather make it more powerful. Automatic service identification can be combined with host-based filtering for more precise filtering that blocks targeted traffic while leaving untargeted traffic unaffected. The first step in avoiding filtering is therefore to have a set of server IPs that have not been added to the filtering blacklist.

The result of the hardware study is the following list of features chosen for inclusion in the adversary models:

- Byte sequence matching on packet contents
- Packet lengths
- Connection duration
- Entropy of contents
- Flow rate in bytes per millisecond

3.2.2 Field Study

After completion of the hardware study, a field study was carried out in which packet capture data was collected in four countries: Burma, Kazahkstan, Jordan, and Nigeria. This study was conducted as part of a larger study on Internet filtering and surveillance technology deployment and policy under a grant-funded project called AGAINST. The research was coordinated by Internews, a non-profit organization that studies Internet freedom worldwide. The research was sponsored by a grant from the US Department of State, Bureau of Democracy, Human Rights, and Labor. The data was collected independently by in-country researchers working for Internews. The role of the author of this dissertation was to create a tool to gather the necessary data and to analyze the results. The result of these efforts was an open-source network measurement tool called CensorProbe and a report, published by Internews, on the state of filtering and surveillance in these countries. Information to download the source to all of the software developed as part of this dissertation research, including CensorProbe, can be found in Appendix D. The data collection methodology was designed to serve a dual purpose of also being useful as a field study for this dissertation.

While CensorProbe collected several different metrics for each country, for the purposes of this dissertation the focus was on the packet capture data. The packet capture data was collected by an automated tool that would record network traffic while driving a dedicated instance of the Firefox web browser to visit a fixed set of HTTP and HTTPS websites. These tests were conducted on a dedicated laptop with CensorProbe pre-installed. In-country researchers were not required to log into any websites. The tests were fully automated and did not require any user interaction other than initiating the test sequence. Therefore, no personal identifying information was collected about the researchers. The only potentially identifying information in the packet capture data was the IP address from which the test was being initiated. Researchers were asked to test on a variety of public networks such as coffee shops and Internet cafes, but were generally given free reign to conduct the tests at the time and place of their choosing. IP addresses were removed in the first stage of analysis, leaving only statistical information about the results of each test.

In order to analyze the data and build adversary models, a set of statistical analysis tools was created called Adversary Lab. Information for downloading the software created in the course of this dissertation, including Adversary Lab, is available in Appendix D. The statistical analysis of the field study data was used to construct models of HTTP and HTTPS traffic. The following questions were investigated through analysis of the data:

- For each chosen feature, what distribution function is a good fit for the data?
- Is the data for each protocol significantly different across different countries?
- Do some features fit better on the HTTP than the HTTPS model or vice versa?

The results from this stage of analysis are presented below. Figure 1 through Figure 18 show graphs of the per-country statistical data for HTTP and HTTPS across the six features.

Duration

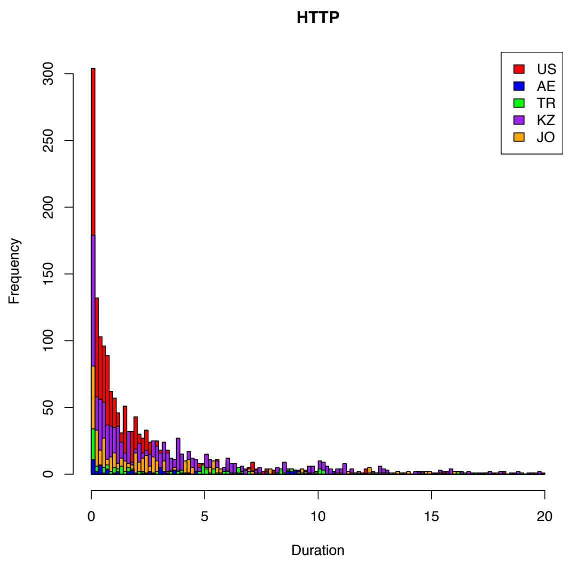

Figure 1: Duration of HTTP connections

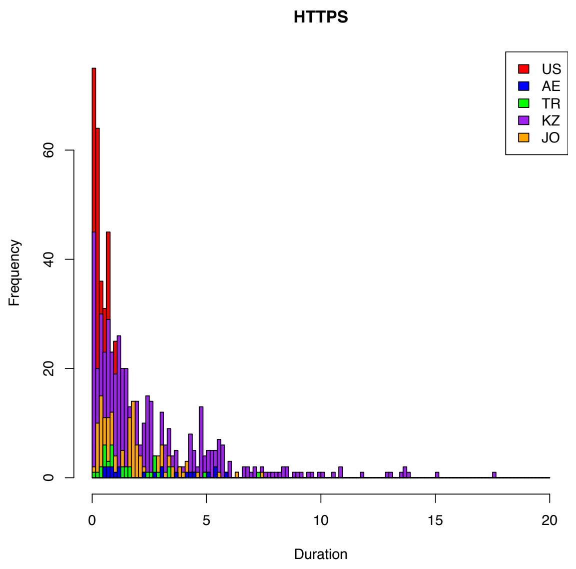

Figure 2: Duration of HTTPS connections

Figure 1 and Figure 2 show histograms of the duration of connections, which is calculated as the interval between the time the first packet and last packet in the connection stream are observed. Packets in both directions were considered when calculating duration, so duration is the one feature that does not have separate incoming and outgoing graphs. Not all values are shown. These graphs show a detailed view of the values from 0 to 20 milliseconds. There is also a relatively flat long tail up to 1000 milliseconds. In these graphs, the x-axis is the duration of the connection in milliseconds. The possible values for the duration and been divided into buckets and the y-axis is the frequency with which values in each bucket have been observed. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution.

Lengths – Outgoing

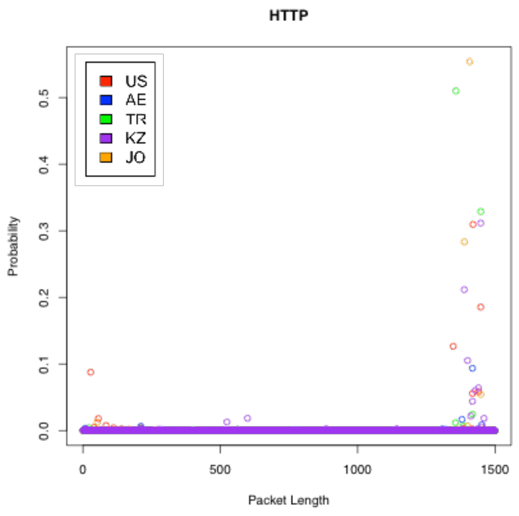

Figure 3: Lengths of outgoing HTTP packets

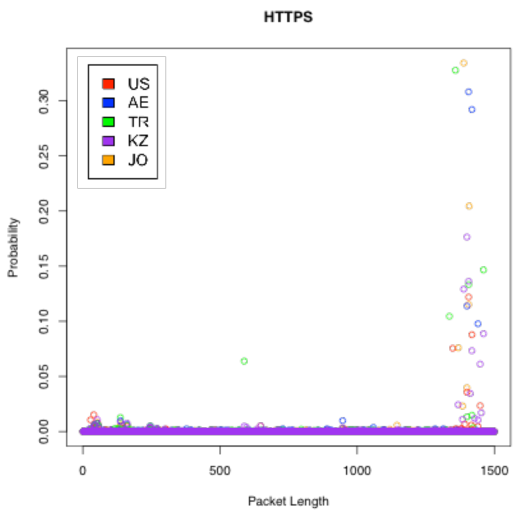

Figure 4: Lengths of outgoing HTTPS packets

Figure 3 and Figure 4 show the observed outgoing packet lengths. As these packets are outgoing from the server, they represent HTTP and HTTPS responses. In these graphs, the x-axis is the observed packet length from 0 to 1500. The y-axis is the empirical probability for each observed packet length. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. They both show a low and fairly evenly distributed probability for most lengths, with a spike towards the high end. This spike is consistent across datasets.

Lengths – Incoming

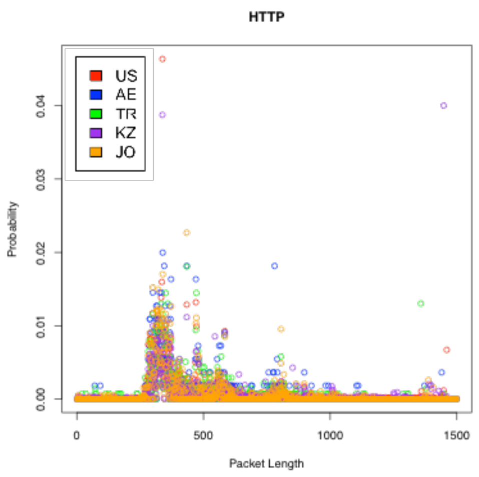

Figure 5: Lengths of incoming HTTP packets

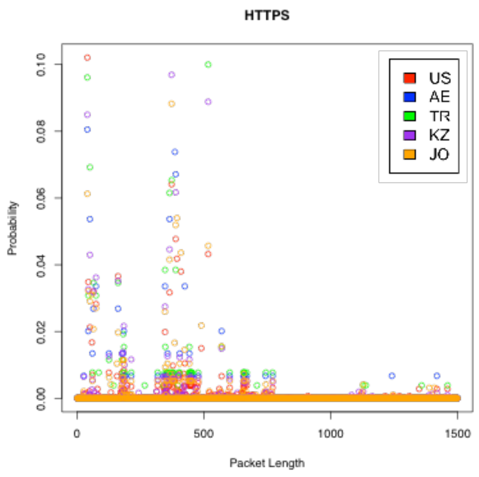

Figure 6: Lengths of incoming HTTPS packets

Figure 5 and Figure 6 show the observed incoming packet lengths. As these packets are coming into the server, they represent HTTP and HTTPS requests. In these graphs, the x-axis is the observed packet length from 0 to 1500. The y-axis is the empirical probability that an observed packet was of that length. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. They both show a fairly even distribution for most lengths, with a number of spikes. The location of these spikes differs between HTTP and HTTPS. For either protocol, the location of the spikes is fairly consistent across datasets.

Entropy – Outgoing

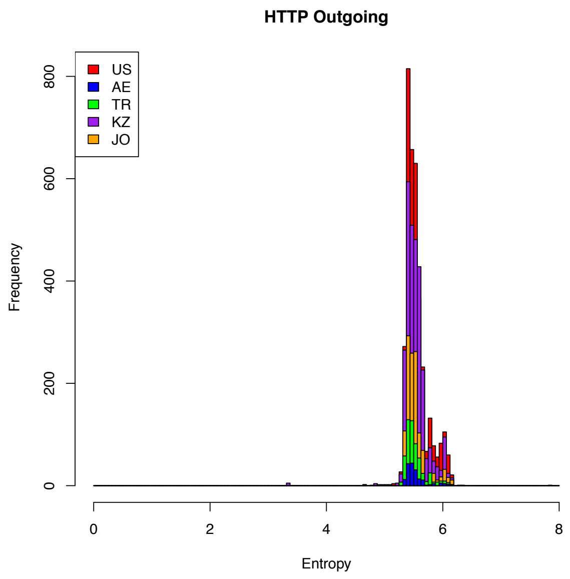

Figure 7: Entropy of outgoing HTTP packets

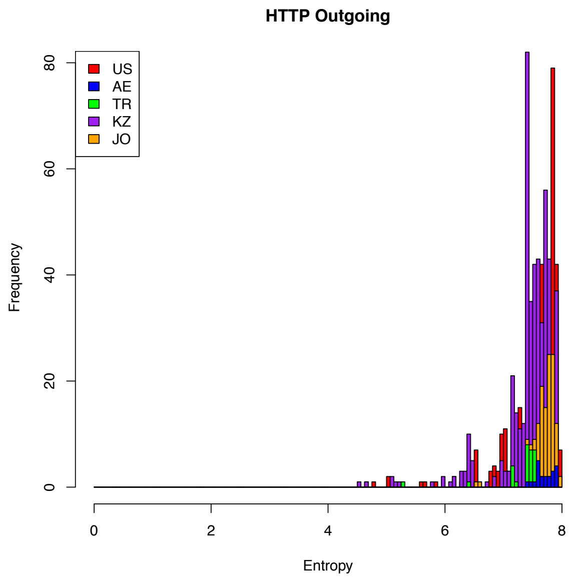

Figure 8: Entropy of outgoing HTTPS packets

Figure 7 and Figure 8 show histograms of the observed outgoing entropy. As these packets are outgoing from the server, they represent HTTP and HTTPS responses. In these graphs, the x-axis is the total first-order entropy of the content of the incoming packets of an observed connection. The possible values for the entropy and been divided into buckets and the y-axis is the frequency with which values in each bucket have been observed. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. The highest value possible for a first-order entropy calculation is 8. HTTPS, being an encrypted protocol, is more frequently observed in buckets close to maximum entropy than HTTP.

Entropy – Incoming

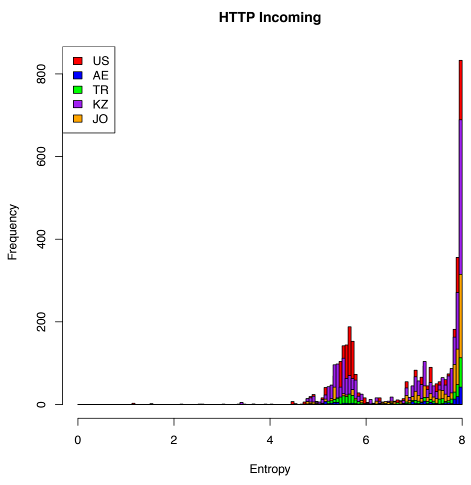

Figure 9: Entropy of incoming HTTP packets

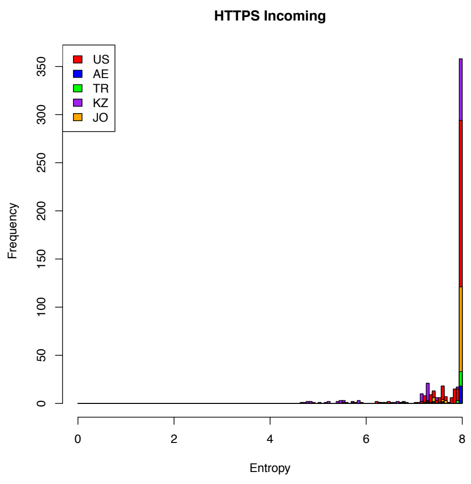

Figure 10: Entropy of incoming HTTPS packets

Figure 9 and Figure 10 show histograms of the observed incoming entropy. As these packets are incoming to the server, they represent HTTP and HTTPS requests. In these graphs, the x-axis is the total first-order entropy of the content of the incoming packets of an observed connection. The possible values for the entropy and been divided into buckets and the y-axis is the frequency with which values in each bucket have been observed. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. The highest value possible for a first-order entropy calculation is 8. HTTPS, being an encrypted protocol, is more frequently observed in buckets close to maximum entropy than HTTP. In contrast to the outgoing entropy values, the incoming entropy values do not asymptotically reach a value of 8, but rather reach up towards 8 and then stop.

Flow – Outgoing

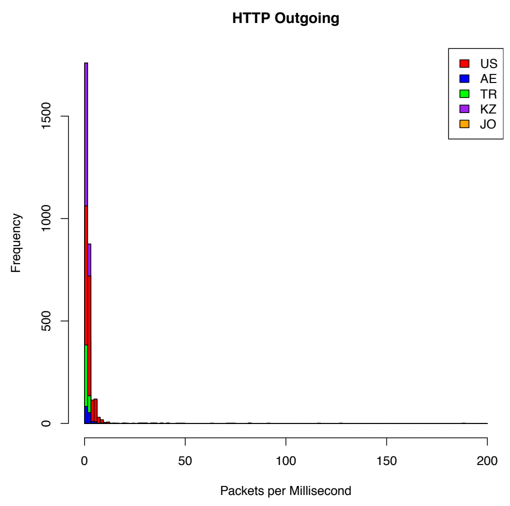

Figure 11: Flow of outgoing HTTP packets

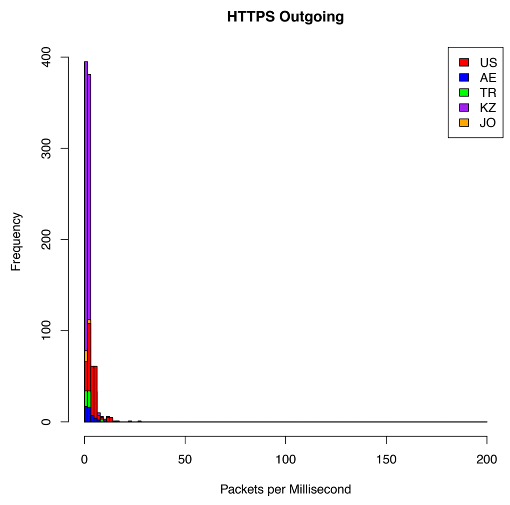

Figure 12: Flow of outgoing HTTPS packets

Figure 11 and Figure 12 show histograms of the observed outgoing packet flow. As these packets are outgoing from the server, they represent HTTP and HTTPS responses. The x-axis is the number of packets observed for each millisecond sample. The possible values for the number of packets per millisecond have been divided into buckets and the y-axis is the frequency with which values in each bucket have been observed. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution.

Flow – Incoming

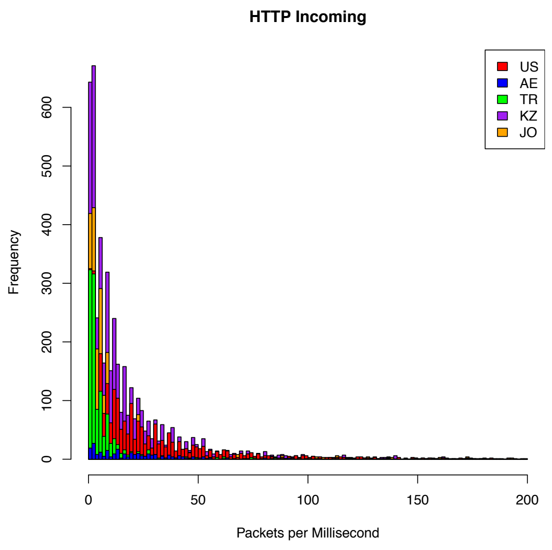

Figure 13: Flow of incoming HTTP packets

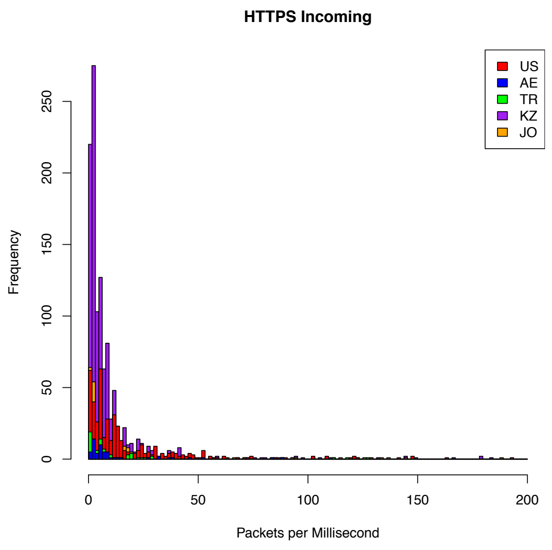

Figure 14: Flow of incoming HTTPS packets

Figure 13 and Figure 14 show histograms of the observed incoming packet flow. As these packets are incoming to the server, they represent HTTP and HTTPS requests. The x-axis is the number of packets observed for each millisecond sample. The possible values for the number of packets per millisecond have been divided into buckets and the y-axis is the frequency with which values in each bucket have been observed. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution.

Content – Outgoing

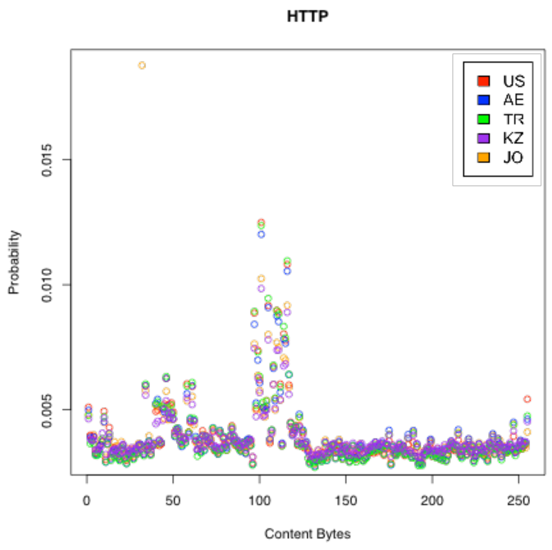

Figure 15: Content of outgoing HTTP packets

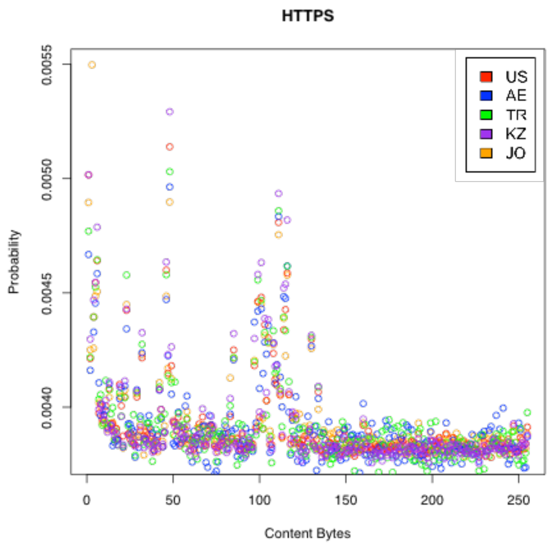

Figure 16: Content of outgoing HTTPS packets

Figure 15 and Figure 16 show the observed first-order distribution of the content of outgoing packets. As these packets are outgoing from the server, they represent HTTP and HTTPS responses. In these graphs, the x-axis is the observed byte value from 0 to 255. The y-axis is the empirical probability of each byte value in the content of outgoing packets. Each color represents a different dataset from a different country. The legend shows the two letter country code for each dataset. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. Probabilities are mostly evenly spread through the available options with spikes around 50 and 100. These spikes are consistent across datasets and occur in both HTTP and HTTPS. HTTPS also has another spike on the low end around 0.

Content – Incoming

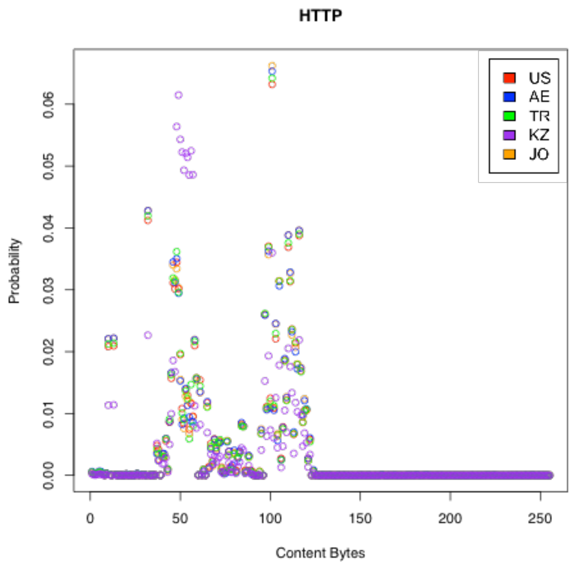

Figure 17: Content of incoming HTTP packets

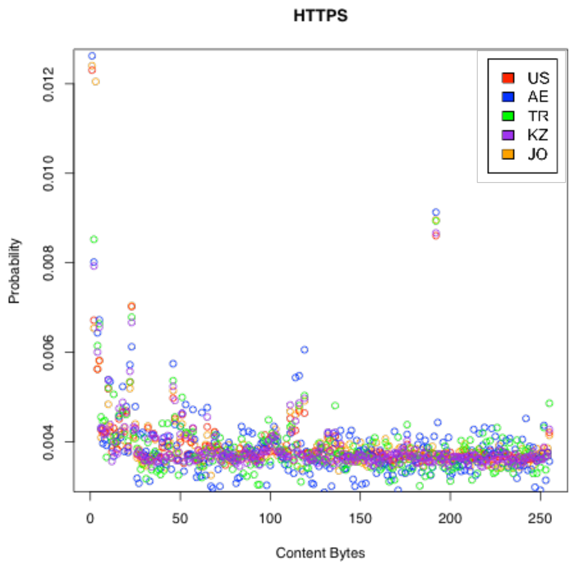

Figure 18: Content of incoming HTTPS packets

Figure 17 and Figure 18 show the observed first-order distribution of the content of incoming packets. As these packets are incoming to the server, they represent HTTP and HTTPS requests. In these graphs, the x-axis is the observed byte value from 0 to 255. The y-axis is the empirical probability of each byte value in the content of incoming packets. Each color represents a different dataset from a different country. The data is represented in this way as the characteristic of the data under consideration is its distribution. As can be seen from these graphs, both HTTP and HTTPS for all datasets follow a similar distribution. Probabilities are mostly evenly spread through the available options with spikes around 50 and 100. For HTTP, most values are near zero, whereas there is more variance for HTTPS. However, both HTTP and HTTPS have spikes around 50 and 100. HTTPS also has another spike on the low end around 0.

Byte Sequence Matching

One feature that is common in filters, but which cannot be represented as a simple probability distribution function, is byte sequences. Specific sequences of bytes starting at fixed offsets, usually somewhere in the first packet, can be used to identify protocols. A circumvention tool must therefore be able to remove byte sequences that identify the traffic as a protocol that the filter seeks to block, and to reproduce byte sequences that the filter requires to be present in traffic that it allows through.

The presence or absence of a byte sequence is an existential property rather than a statistical one. For each connection, either the byte sequence is present or it is absent. For each filtering rule, either a specific byte sequence is required, or it is forbidden, or it is ignored. So while the filters do not use statistical rules when employing byte sequence matching, a statistical analysis can still be used to find which byte sequences are most probably used in filtering rules. A byte sequence extractor was developed that finds probable byte sequence that could be used to effectively identify the protocol of observed traffic. Illustrations 1 through 4 show byte-sequence probability information for both incoming and outgoing directions for HTTP and HTTPS.

HTTP - Incoming

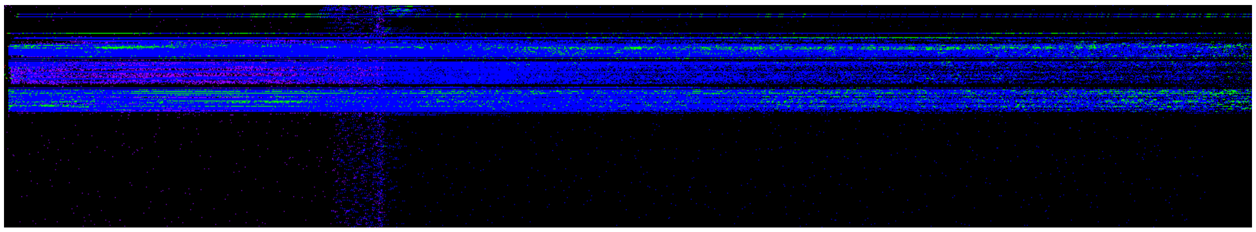

Illustration 1: Positional probability of byte values in incoming HTTP packets

HTTP – Outgoing

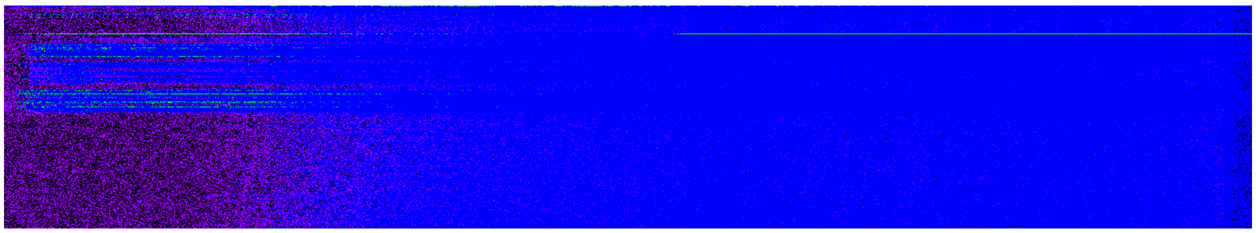

Illustration 2: Positional probability of byte values in outgoing HTTP packets

HTTPS – Incoming

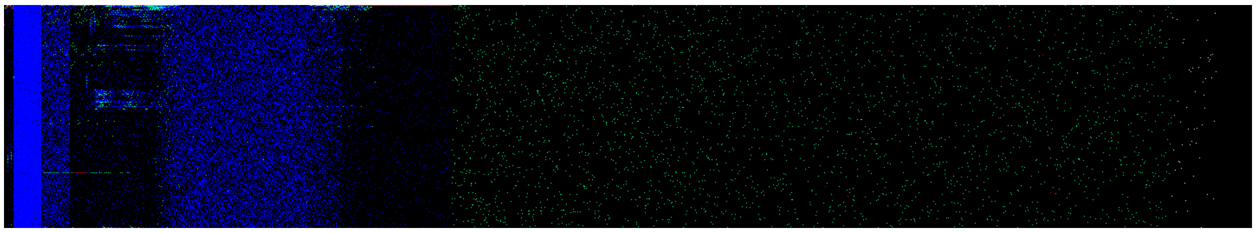

Illustration 3: Positional probability of byte values in incoming HTTPS packets

HTTPS - Outgoing

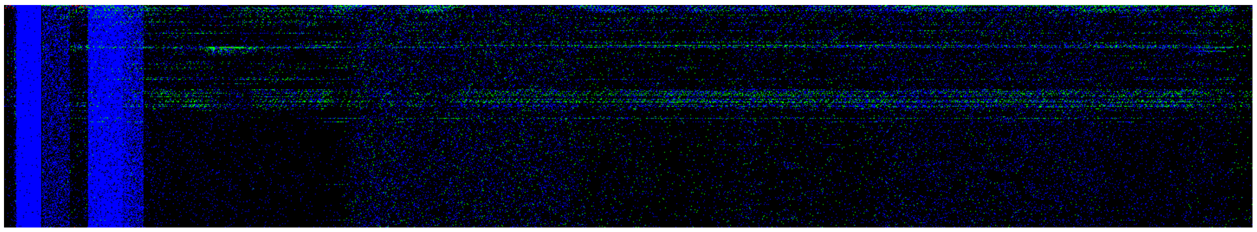

Illustration 4: Positional probability of byte values in outgoing HTTPS packets

Expanded versions of these images that give more detail are available in Appendix B.

Illustrations 1 through 4 are observed probability heat maps of the content of the observed traffic. Each column of pixels represented a fixed offset within the first packet from 0 on the left to 1440 on the right. Each row shows a particular byte value from 0 on the top to 255 on the bottom. The color of the pixel represents the probability that the byte value for that row was observed at the offset for that column. The color mapping for probabilities is chosen according the following scheme:

- Probability of 0 – Black
- Probability less than 0.00039 – Purple
- Probability between 0.00039 and 0.039 – Blue
- Probability between 0.039 and 0.5 – Green
- Probability between 0.5 and 1 - Red
- Probability of 1 – White

The blue range is the range of probabilities within one order of magnitude of uniformly random. With 256 possible values, equally probable values occur with a probability of 1/256 or 0.0039. While an exactly uniformly random probability is unlikely to occur due to the approximations involved in sampling, values within an order of magnitude can be considered approximately equal to a uniform distribution. Therefore, a range between 0.00039 and 0.39 was chosen as the range for approximately uniformly random values. The color purple represents values that are over an order of magnitude less likely to occur than values in the uniformly random range. The color green represents values that are over an order of magnitude less likely to occur than values in the uniformly random range. Values that are red occur more than 50% of the time, making them good candidates for byte sequences that can be used to identify traffic using the given protocol.

When interpreting the data, it is important to remember that this is a visualization of the samples and that certain artifacts may show up in the visualization that are the result of sampling. Not all packets are the same length, so toward the larger offset values less data is available. This causes more black pixels to appear as the offset increases. Additionally, there is a shift in the color towards the red end of the spectrum when there is less available data since the values that are observed have higher observed probability. There is also a shift towards red for protocols in which less data was captured for analysis. When observing a uniformly random protocol, the expected result is for it to begin as white with the first observation and move toward blue as more data is observed. Therefore, it is inappropriate to compare colors across images because different amounts of data were used to build each image.

These images give an overall impression of the distribution of content probabilities based on offset into the first packet. For the purpose of finding identifying byte sequences, it is useful to zoom in on the subset of offsets that have red pixels. Illustrations 5 through 8 have been cropped and zoomed to show the first few bytes of each original image.

HTTP – Incoming

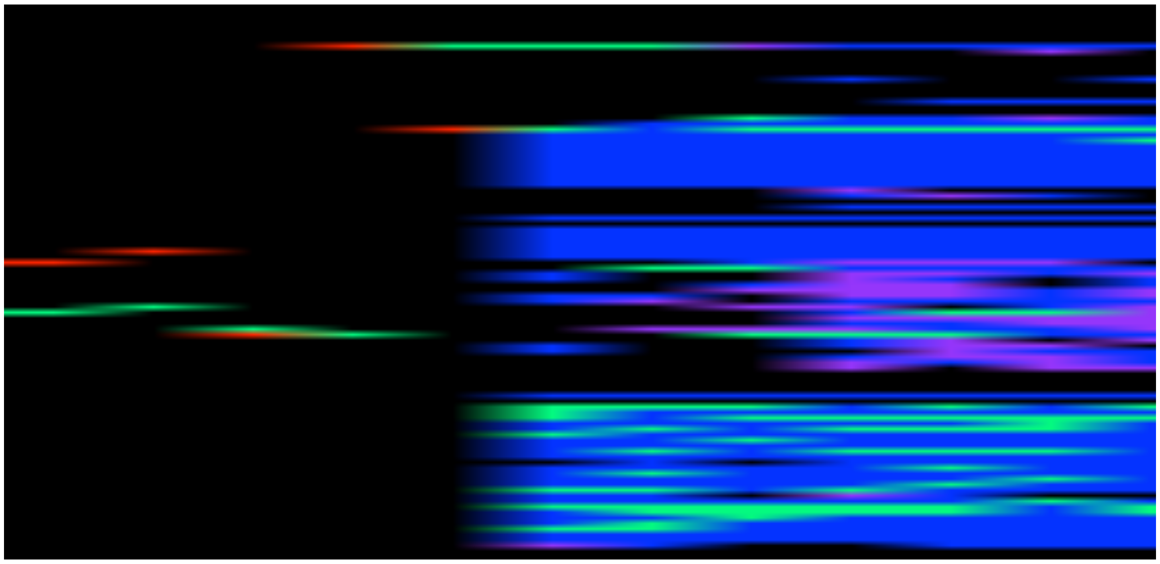

Illustration 5: Close up of positional probability in incoming HTTP packets

HTTP - Outgoing

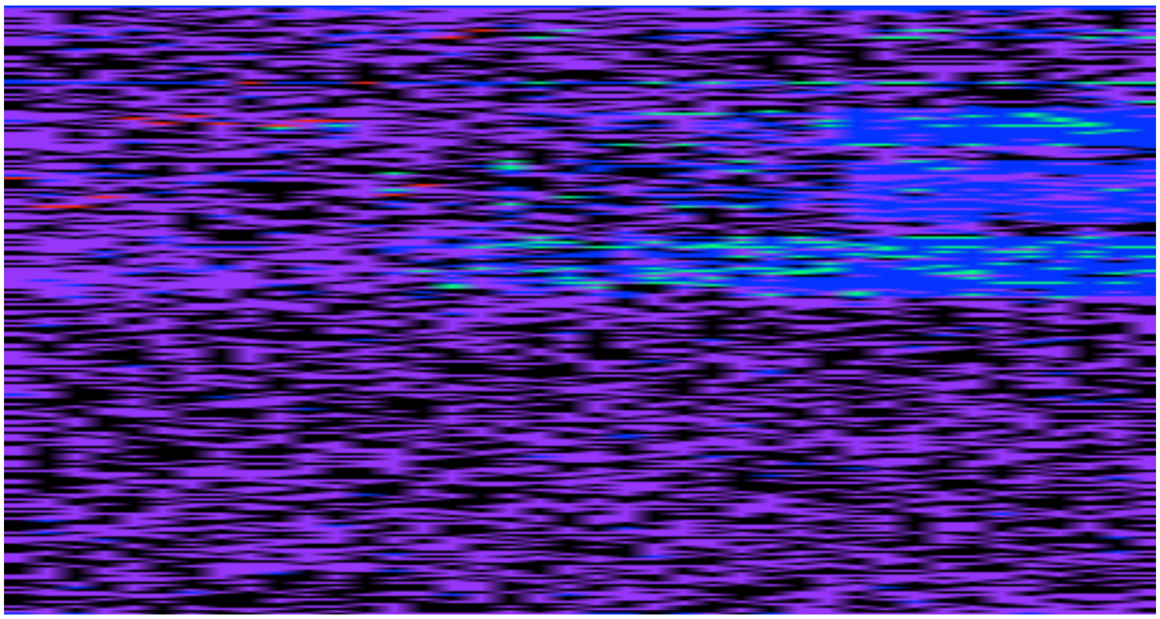

Illustration 6: Close up of positional probability in outgoing HTTP packets

HTTPS – Incoming

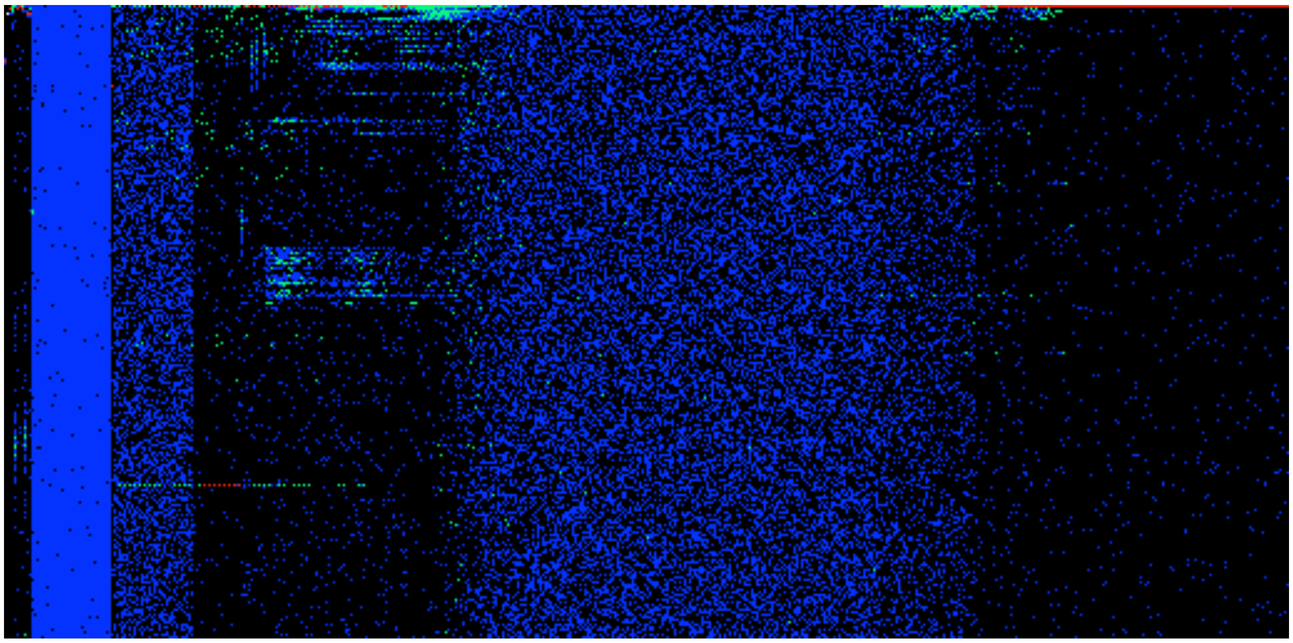

Illustration 7: Close up of positional probability in incoming HTTPS packets

HTTPS - Outgoing

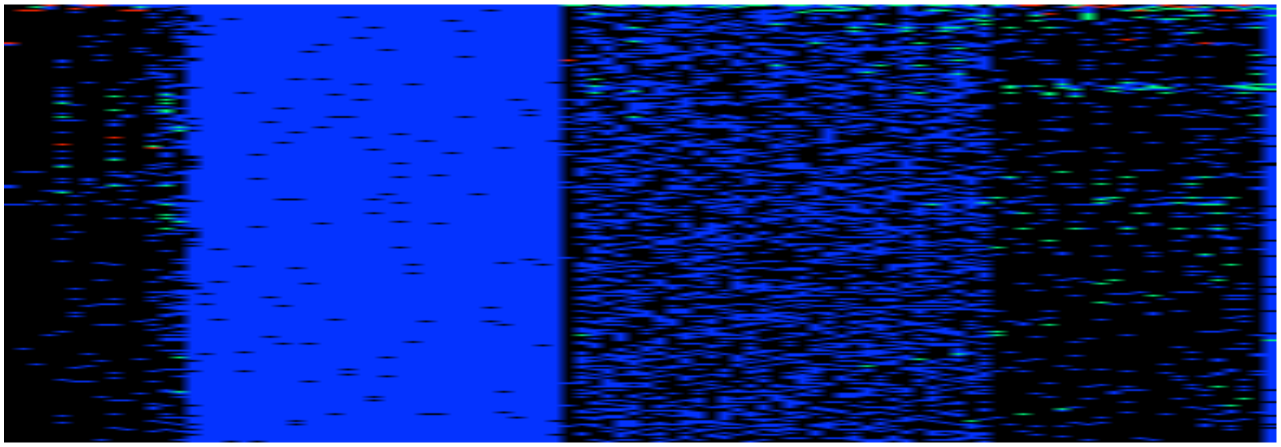

Illustration 8: Close up of positional probability in outgoing HTTPS packets

What each image shows is that while the length and specific offsets for red pixels differ by protocol and direction, they all occur in clusters at the beginning of the each observed packet.

Based on this distribution of high probability bytes, a *byte sequence extractor* was created. It takes into account the following observations:

All observed protocols had a byte value that occurred with greater than 0.5 observed probability at offset 0.

By definition, only one byte value can occur with greater than 0.5 probability at a given offset.

The byte sequence extractor therefore uses the following algorithm:

- Start with a set of all connections and offset 0
- Loop
    - For the given offset, find the byte value that has greater than 0.5 probability at the given offset. If it exists then record this value and continue, otherwise terminate
    - Take the subset of connections which have the given byte value at the given offset, discarding the rest
    - If at least two connections remain in the set then continue, otherwise terminate
    - With the given set of connections, calculate a new probability distribution using only those connections
    - Increment the offset by 1 and repeat the loop

The result of this algorithm is to produce the most probable byte sequence that starts with the most probable first byte. It terminates when there is not a clearly most probably byte for the next byte in the sequence. By dividing the connections into progressively smaller subsets, the algorithm can work efficiently with a small collection of data rather than building a very large probability model of the entire space of byte sequence probabilities. The limitation of this algorithm is that it found exactly one (or zero) byte sequence per set of connections analyzed. It can also only identify contiguous sequences that start with the first byte in the packet. In future research, the algorithm could be extended to find multiple disconnected sequences.

Adversary Models

The HTTP and HTTPS model data was then used to construct a sample adversary that classifies traffic as either HTTP or HTTPS. To create this adversary model, a 2-fold cross-validation approach was used. All HTTP from all countries was pooled and the HTTPS data was similarly pooled to create two master datasets. The packet capture data from each test was broken into individual connections and all the connections for each protocol were put into the master pools. The master pools were then randomly partitioned into training and test data sets. The training data set was used to build an HTTP model and an HTTPS model. These models were then used to classify the HTTP and HTTPS data in the test dataset. For each connection in the test dataset, the predictions were generated from the trained HTTP and HTTPS models and compared to the empirical data from the connection. For each feature, a root-mean-square error (RMSE) was calculated as a goodness-of-fit metric. A score was then calculated for the feature. If the RMSE for HTTP was less than for HTTPS, a score of 1 was given. If the RMSE for HTTP was greater than for HTTPS, a score of -1 was given. If the RMSE for both were the same, a score of 0 was given. All of the feature scores were then added to produce a final score ranging from 6 to -6. If the final score was greater than 0, then the classification for this connection was HTTP. If the final score was less than 0, then the classification for this connection was HTTPS. If the final score was 0, then the classification for this connection was Unknown. The classification was then compared to the known correct answer. True positives, false positives, true negatives, false negatives, and unknown classifications were recorded, as well as the overall accuracy in terms of correct and incorrect classifications.

Adversary Model – Iteration 1

In order to best match the empirical distributions shown in the above graphs, the following distributions functions were selected:

- Connection duration - Exponential
- Packet lengths - Multinomial
- Entropy of contents - Normal
- Flow rate in bytes per millisecond – Poisson
- Byte distribution of contents - Multinomial

Figure 19 through Figure 26 show the distribution of correct, incorrect, and unknown classifications, both across features and as combined to form a final score.

Accuracy of Duration Classifier

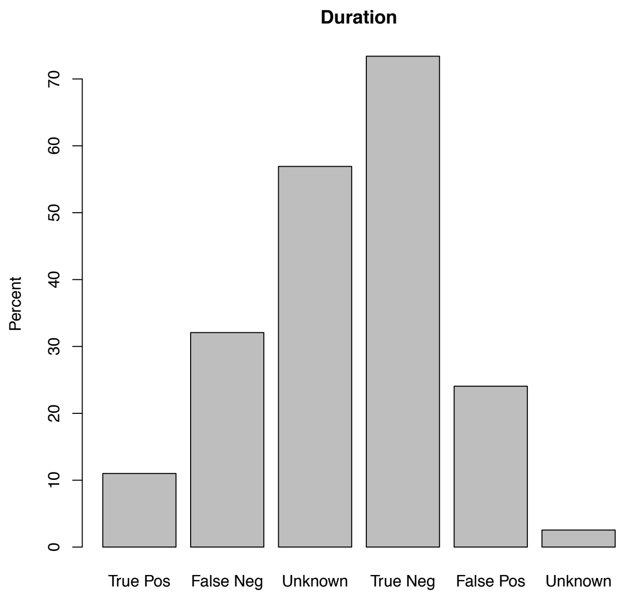

Figure 19: Accuracy of duration classifier for test adversary iteration 1

Accuracy of Content Classifier

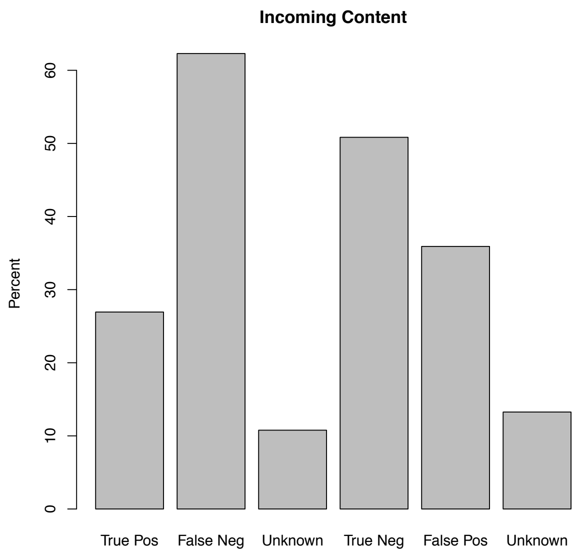

Figure 20: Accuracy of incoming content classifier for first iteration of test adversary

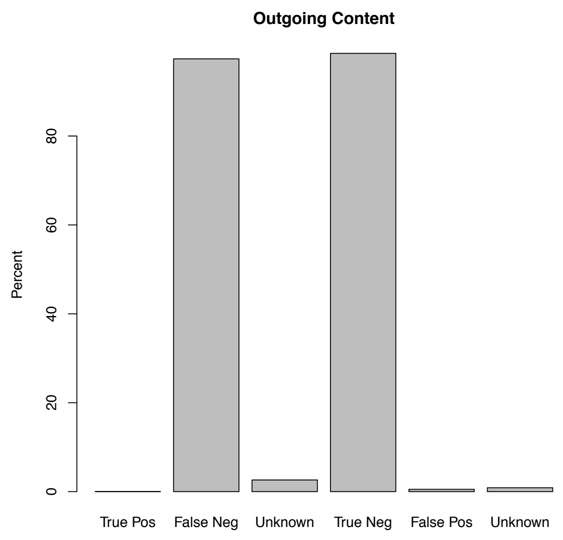

Figure 21: Accuracy of outgoing content classifier for first iteration of test adversary

Accuracy of Entropy Classifier

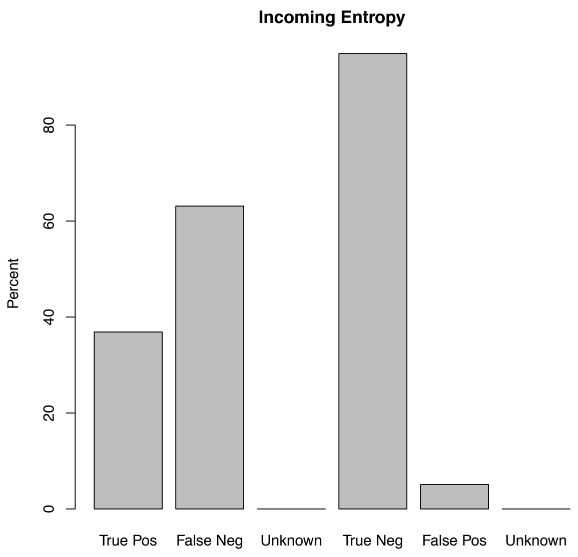

Figure 22: Accuracy of incoming entropy classifier for first iteration of test adversary

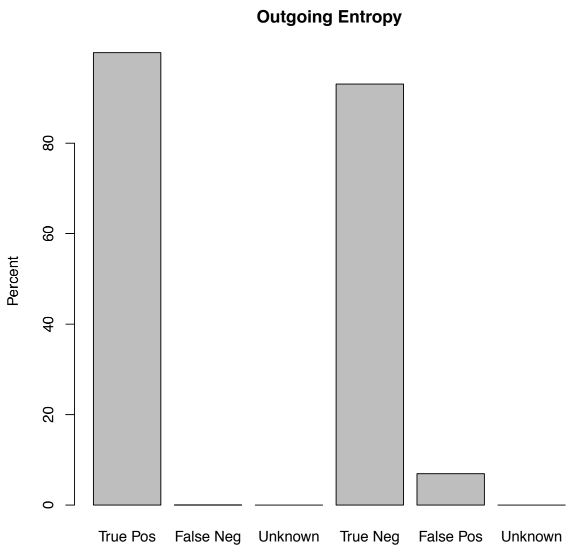

Figure 22: Accuracy of incoming entropy classifier for first iteration of test adversary

Accuracy of Flow Classifier

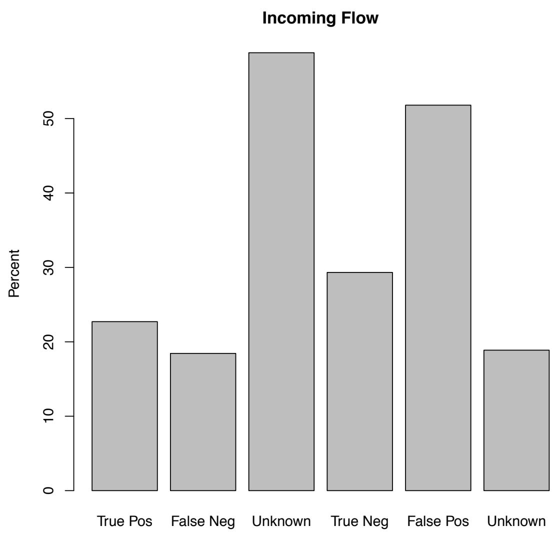

Figure 23: Accuracy of incoming flow classifier for test adversary iteration 1

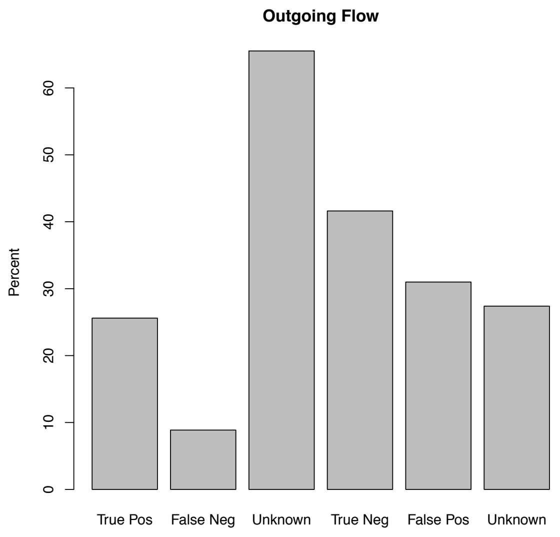

Figure 24: Accuracy of outgoing flow classifier for test adversary iteration 1

Accuracy of Length Classifier

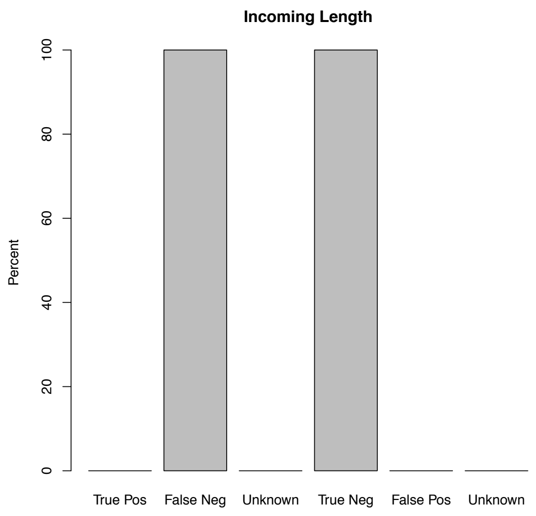

Figure 25: Accuracy of incoming length classifier for test adversary iteration 1

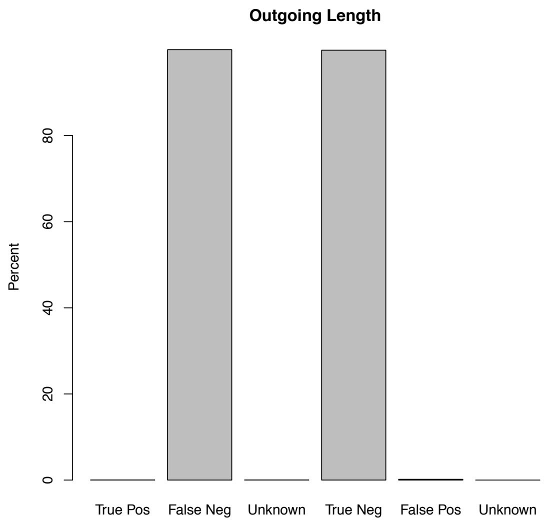

Figure 25: Accuracy of incoming length classifier for test adversary iteration 1

Total Accuracy of Classifiers for Iteration 1

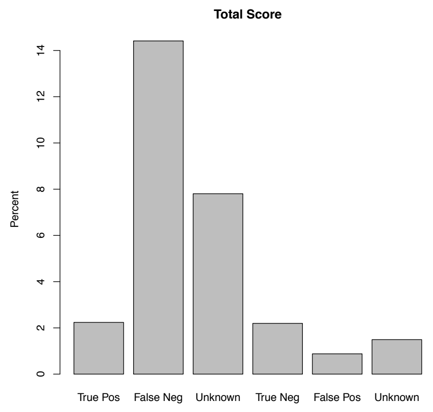

Figure 26: Total accuracy of classifiers for test adversary iteration 1

Adversary Models - Iteration 2

Given the poor performance of the adversary using the distribution functions from Iteration 1, additional iterations were made to refine the models.

For the duration feature, 4 models were tested. The exponential function was compared against a gamma. These two distributions, exponential and gamma, were fit to the original data as well as to a "clean" variant in which all 0 values were removed, the values were sorted, and the middle 95% of values were extracted while the 5% of high and low outliers were discarded. The goal of this cleaning process was to achieve a better fit by discarding values that might decrease the goodness-of-fit. The RMSE goodness-of-fit scores for the four duration fitting tests are shown in Table 1.

Test	**Incoming**	**Outgoing**	**Total**
exponential	**2534**	**2829**	**5363**
exponential-clean	**4554**	**1569**	**6124**
gamma	**3358**	**2937**	**6295**
gamma-clean	**5059**	**1717**	**6776**

Table 1: Comparison of RMSE sores for duration fitting tests

Based on these results, the original exponential using the original data has the lowest total RMSE, making it the best overall fit. Looking at the graphs' duration, it is apparent that there are values that should clearly indicate the set from which the duration was drawn. However, exponential distributions cluster most of their density at lower values, and these values are very similar between the two sets. Therefore, an alternative method for classification was tried, which tunes the adversary to reduce false positives and false negatives. In this alternative method the value to be tested was used to index a probability density value in the curve of each set's fitted distribution function. The higher-probability density value was considered to be the classification. When this method was applied to the exponential curve fit to the data, it was discovered that HTTPS almost always had higher probability densities than HTTP when testing against both HTTP and HTTPS data. However, the probability densities were even higher when the test data was in fact HTTPS. Therefore, an iterative algorithm was used to find the cutoff density value that differentiated HTTP and HTTPS traffic such that the percentage of errors was the same for both HTTP and HTTPS. An offset of 0.138 was found to be optimal. A comparison of the original method and the new method is shown in Figures 27 and 28.

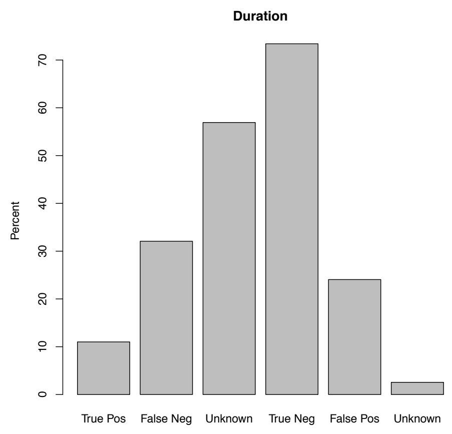

Figure 27: Accuracy of duration classifier for test adversary iteration 1

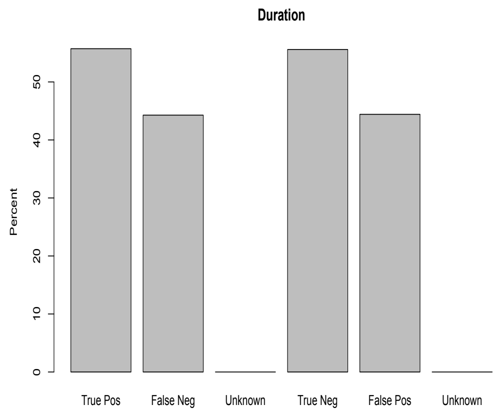

Figure 28: Accuracy of duration classifier for test adversary iteration 2

As is evident from Figures 27 and 28, the new method has better accuracy in terms of correctly reporting true positives and true negatives with greater probability than false positives and false negatives. However, the new method does not allow for any tests to have unknown results, leading to an increase in the number of false positives and false negatives, which previously would be classified as unknown. Overall, the new approach produces better results as it has an overall improved accuracy.

Examining the other features revealed that some were not good discriminators between HTTP and HTTPS. Packet length, content distribution, and packet timing were sufficiently similar between HTTP and HTTPS that the accuracy of discriminators using these features was low. An example of this is seen in Figures 29 and 30, displaying the models for length. Though they are not identical, there is too much overlap to create an accurate discriminator. This is not entirely surprising inasmuch as HTTP and HTTPS are related protocols and so are likely to show some amount of similarity. Since HTTPS is HTTP that has been encrypted using SSL, some of the characteristics of the underlying HTTP protocol might show through in the HTTPS traffic.

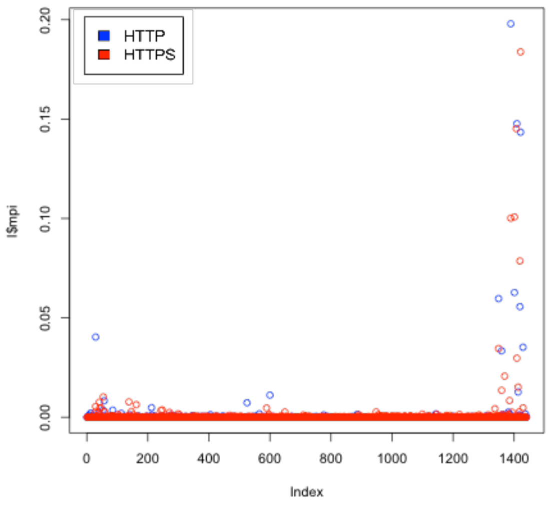

Figure 29: Packet length distribution of incoming packets

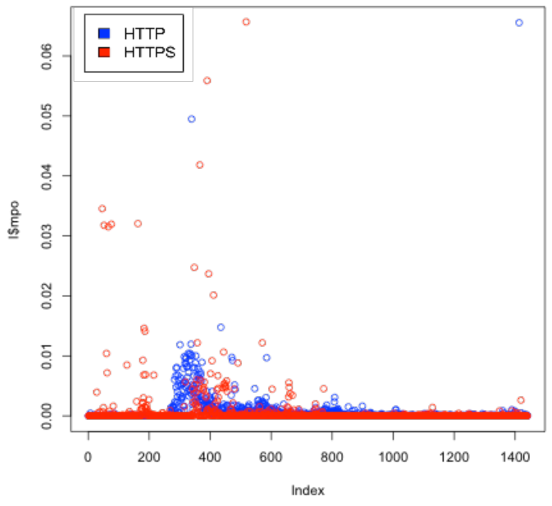

Figure 30: Packet length distribution of outgoing packets

Perhaps the most surprising result was the content distribution, as HTTP is unencrypted and HTTPS is encrypted. Indeed, there are notable differences in the content distribution when analyzing a large amount of data. However, the shapes of the distribution are similar. This makes it difficult to determine, when looking at a small amount of data such as a single connection, the distribution that most likely produced it. This is seen in Figures 31 and 32.

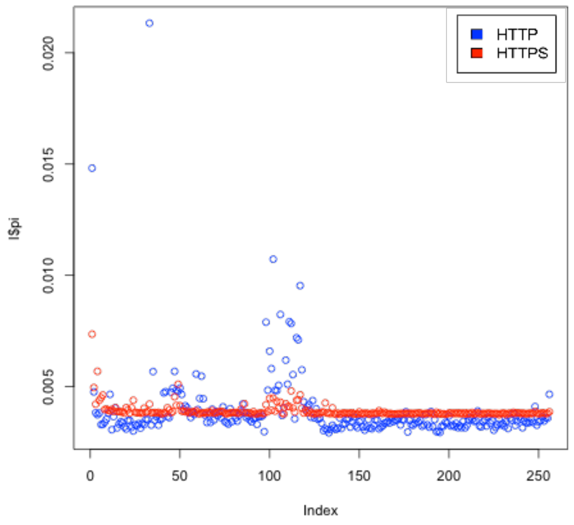

Figure 31: Distribution of byte values in the content of incoming packets

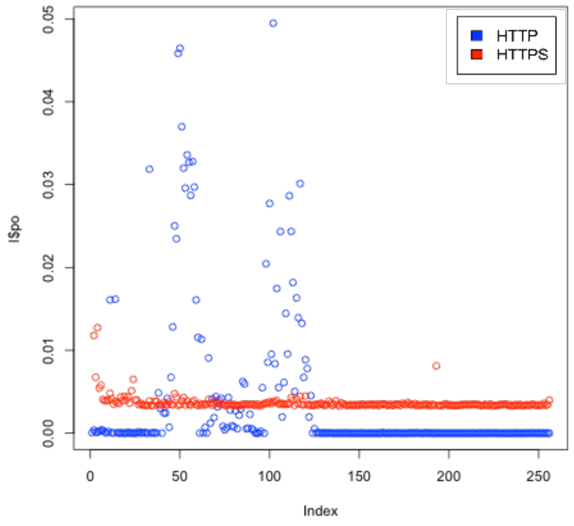

Figure 32: Distribution of byte values in the content of outgoing packets

Two features proved to be much better discriminators for HTTP-HTTPS: entropy and byte sequence matching. Entropy is interesting because it is derived from the content distribution, but it also provided better accuracy than direct comparsion of the content distribution. This is because, unlike a model based directly on the content distribution, it does not consider the specific shape but only the relative flatness or unevenness of the distribution. As is evident in the graphs above, the two protocols show a different amount of flatness in their distributions. Figures 33 and 34 show the entropy models derived from the data.

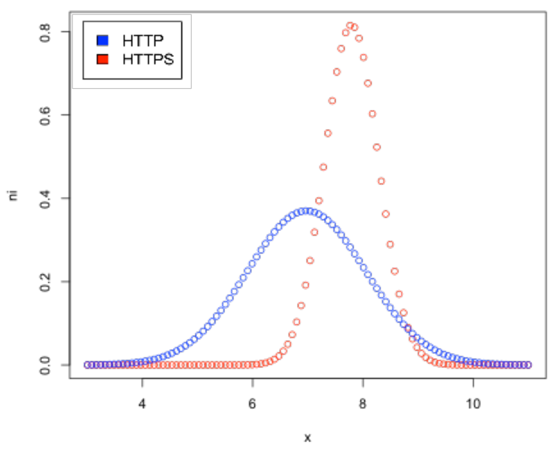

Figure 33: Comparison of models of incoming entropy for HTTP and HTTPS

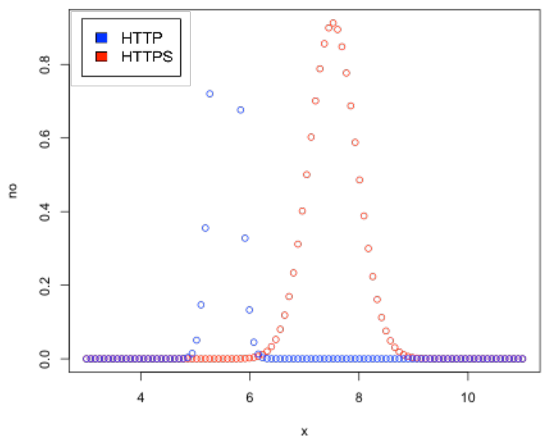

Figure 34: Comparison of models of outgoing entropy for HTTP and HTTPS

Although both incoming and outgoing models showed differences between the protocols, the overlap on the incoming models was too large to provide good discrimination. In contrast, the outgoing model shows little overlap between the two distributions. This allowed for the creation of an accurate discriminator using only outgoing entropy.

The other feature that worked well was byte sequence matching. Scoring was somewhat different for the byte sequence matching inasmuch as this feature does not represent a statistical distribution, but instead an existential quality that the connection either does or does not possess. Each tested connection either did or did not contain the specified byte sequence at the particular offset for the model. The scoring algorithm therefore only produces values of 1 for a match and 0 for no match. In order to fit both HTTP and HTTPS on one graph, the HTTPS results were multipled by -1, so they range from 0 (no match) to -1 (match). The byte sequence matching produced no false positives and no false negatives. All results were a true positive (HTTP match), a true negative

(HTTPS match), or an unknown classification (no match). Figures 35 and 36 show the results of the byte sequence matching.

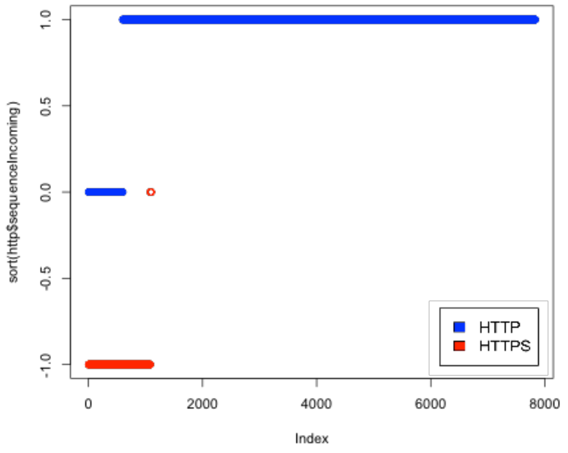

Figure 35: Comparison of results of incoming byte sequence matching

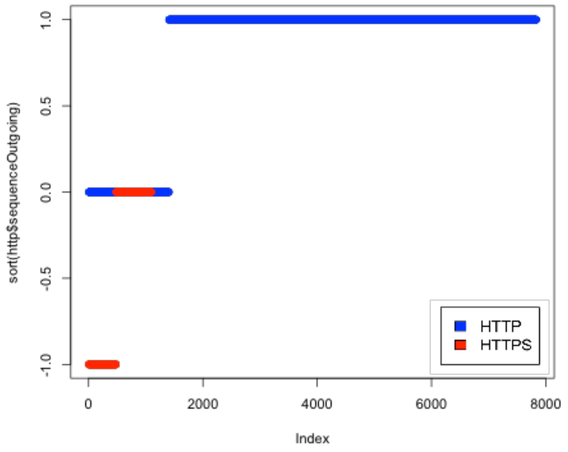

Figure 36: Comparison of results of outgoing byte sequence matching

Due to the lack of false positives and false negatives, the byte sequence matching was an excellent discriminator. The only drawback is that it was not able to classify all of the connections. Fortunately, it can be combined with the outgoing entropy to produce a final adversary model. Figures 37 and 38 show the combined scores for an adversary using outgoing entropy and both incoming and outgoing byte sequence matching.

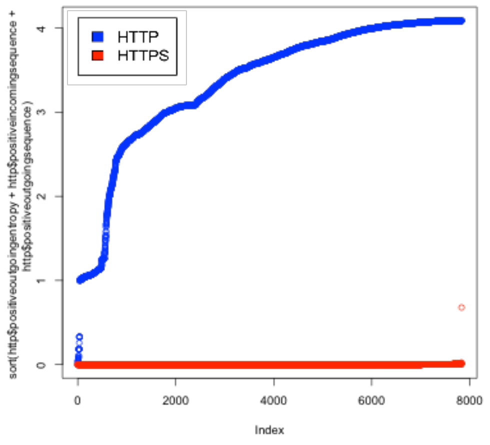

Figure 37: HTTP matching scores for combined entropy and byte sequences

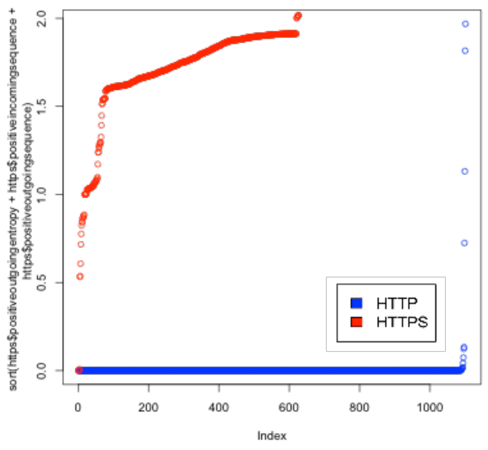

Figure 38: HTTPS matching scores for combined entropy and byte sequences

As is evident from the graphs, the combined features produced very good discrimination between HTTP and HTTPS. Figure 39 shows the final evaluation of this adversary model:

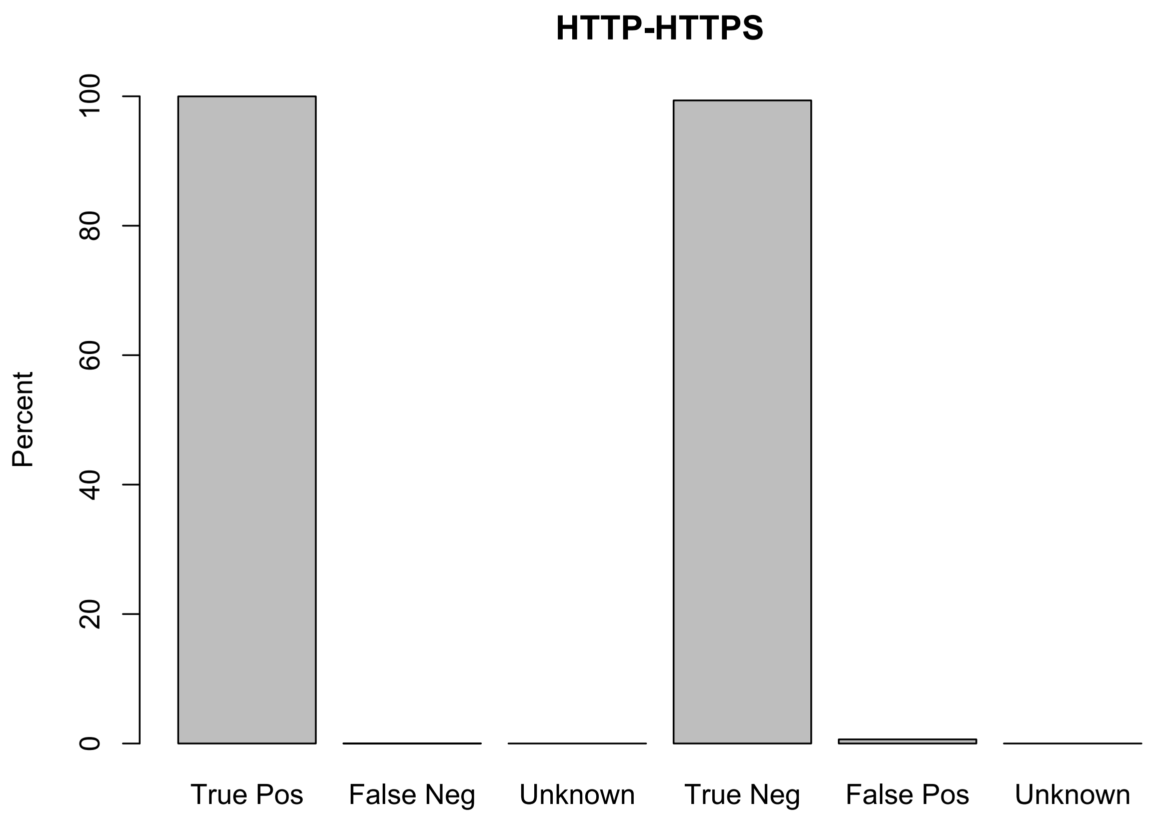

Figure 39: Total accuracy for test adversary iteration 2

The adversary has over 99% accuracy for correctly identifying both HTTP and HTTPS.

This final adversary concludes the initial work in exploring feature-based statistical modeling of filters. The HTTP and HTTPS samples collected during the field study were used to create an adversary model that had 99% accuracy at distinguishing between HTTP and HTTPS. During this process it was demonstrated that an important part of the adversary modeling process was to choose those features which are good discriminators for the specified set of protocols being examined.

3.3 ADVERSARIES

After establishing through the HTTP-HTTPS proof of concept adversary what the probability distributions for each feature should be, adversaries were created for use in the evaluation of the Dust engine. As an outcome of the value-sensitive design reflection process, the chosen protocols for encoding using Dust were HTTP, HTTPS, and RTSP. Therefore, adversaries were created to differentiate traffic using these protocols. Six adversaries were created using traffic for these protocols. First, three adversaries were created as baselines. The baseline adversaries were HTTP-HTTP, HTTPS-HTTPS, and RTSP-RTSP. With the baseline adversaries, the same protocol was used for both categories. However, the same data was not always used for both categories. The datasets were once again divided into two sets, which we can call X and Y. Different combinations of data were used for training and testing. For instance, in one variant the HTTP X dataset was using for training one side and the HTTP Y dataset was used for training the other side. In other variations, the same dataset was used for training both sides. The purpose of this set of adversaries is to establish a baseline for accuracy of the classifier when no interesting signal exists. These baseline adversaries are evaluated in the evaluation section. The second set of three adversaries differentiate between the combinations of the three different protocols. The adversaries are as follows: HTTP-HTTPS, HTTPS-RTSP, and HTTP-RTSP. These adversaries were similarly built in several variations, using combinations of data from the X and Y subset for each protocol. The second set of adversaries is also evaluated in the evaluation section. Additionally, these three adversaries were used to evaluate Dust. In order to evaluate Dust, the same adversaries were used to classify a mixture of traffic containing both traffic for a protocol observed in the field and traffic encoded using Dust. For instance, the HTTP-HTTPS adversary was used to classify a mixture DustHTTP traffic and HTTPS traffic. This evaluation is discussed in detail in the evaluation section. Finally, a new set of adversaries was created specifically to classify Dust encoded traffic. Three adversaries of this type were created: DustHTTP-HTTP, DustHTTPS-HTTPS, and DustRTSP-RTSP. The purpose in creating these adversaries was to see if new adversaries trained on Dust traffic could be created that could differentiate between Dust encoded traffic and the traffic it was attempting to mimic. All of the different tests for these adversaries can be seen in Appendix C, and the results are summarized in the evaluation section.

3.4 Conclusion

The work of model building started by looking at filtering hardware to determine a semantic model detailing what features of network traffic are used by filters to classify traffic. Using this set of features, a field study was done in 4 countries collecting network traffic and analyzing the features of this traffic. A prototype adversary was then created using the analyzed data to differentiate between HTTP and HTTPS traffic. This adversary was tuned until it achieved a 99% accuracy in classification. The process of tuning helped to decide on the best probability distribution to use for each feature in order to optimize goodness-of-fit to the observed data as well as accuracy of the classifier. Following the proof-of-concept adversary, new adversaries were created using the protocol selected in the value-sensitive design reflection phase of the project. Adversaries were created to differentiate traffic between these protocols. These adversaries were also used to evaluate the performance of the Dust engine, as is discussed in the evaluation section.

The building of a collection of adversary models could be the work of a lifetime. Every filtering device is a unique adversary that can be configured into numerous different configurations. The research literature is also always developing new hypothetical adversaries. The plan in this research project is to build a standard methodology for creating computational models of adversaries that can be compared to each other and tested against various obfuscating encodings. Ideally, this work will serve as the starting point for the creation of many adversary models by many different researchers.

4 Design of the Engine

4.1 Introduction

Having completed the value-sensitive design process and the creation of the models, the next phase of research was to use those models to encode traffic and send it through simulated adversaries. The reasoning was that simulated adversaries would allow encoded traffic through that would be blocked if it were not encoded. Generating encodings from filter models required the creation of a traffic-generation engine that takes as input the traffic to be encoded as well as a statistical model of allowed traffic and produces as output traffic that conforms to the allowed traffic model. This transformation must be reversible so that the original traffic can be recovered after it has passed through the filter. This engine is called Dust and constitutes the primary engineering contribution of the dissertation and to the literature of unobservable communication. Unlike much of the previous work in the field of unobservable communication, Dust is not a single obfuscated protocol, but a polymorphic protocol engine capable of generating traffic to match any protocol for which there is a model. In the spectrum of unobservable communication approaches, it is most closely related to Format-Transforming Encryption (FTE) [8]. However, unlike FTE, it does not require a formal grammar to be specified for the protocol. Instead, statistical models are built by sampling observed traffic. Therefore, Dust encodings can be built for undocumented and proprietary protocols for which no formal specification exists. To distinguish the approach used in Dust from earlier network protocol obfuscation techniques, the technique used in Dust is referred to as *polymorphic protocol shapeshifting*. This is a new term, originating in this dissertation, to describe approaches such as FTE and Dust that encode traffic in a way that is configurable to yield a multitude of different encodings with the same software engine. *Polymorphic protocol shapeshifting* advances the state of the field from earlier network protocol obfuscation techniques that encoded traffic according to a static configuration such that all traffic generated by a particular obfuscation system would have similar characteristics. The Dust software is open source and available for use by researchers and tool builders. Information for downloading the software created in the course of this dissertation, including Dust, is included in Appendix D. The name "Dust" was inspired by the system described in section 2.3, "Design Concept 1". In this system, the transport protocol utilized conversations consisting of single UDP packets. The original idea was that this approach would not give the filter enough information to classify the traffic. So the shapeshifted packets would be difficult to block, much as it would be difficult to filter all of the dust out of the air. This turned out to be a false assumption, as network filters actually sometimes look at only the first packets on a conversation for purposes of classifying the conversation. While the final architecture ended up being very different, the name remained.

4.1.1 Use Case

The use case for Dust assumes that there is a network connected to the Internet through a filtering device that selectively blocks some traffic while allowing other traffic through. It is also assumed that there is an unfiltered, out-of-band channel of communication available to users of this network. This is an assumption that is shared by circumvention tools in general for, in order to use software tools to circumvent filtering, users must first obtain these tools via some channel. Dust uses this channel not just to distribute the software, but also to bootstrap a secure obfuscated connection.

4.1.2 Packet Filtering Techniques

The packet-filtering techniques currently in use by deployed filtering hardware devices attempt to keep as little state as possible in order to be scalable. Filtering can happen either at the individual packet level by dropping blocked packets or at the stream level by injecting reset packets to end the connection. For stream-level filtering it is common to sample only the initial packets of the stream. Filters usually do not keep persistent state about streams. Techniques for defeating filtering can therefore concentrate on sending only packets that do not trigger blocking rules.

There are two general classifications of techniques for determining whether a packet is blocked. *Shallow packet inspection* uses just the headers of the packet. This is less expensive because the headers need to be examined anyway in order to route the packet. *DPI* examines the packet contents as well as the headers. *DPI* used to be too expensive to be practical, but it is now in widespread use by some filters. *Shallow packet inspection* techniques are being used for marking packets in several ways: source IP and port, destination IP and port, and packet length. *DPI* techniques are being used for marking packets with byte sequence matching, particularly for connection headers and handshakes of known protocols, and entropy of packet contents.

4.1.3 How Dust Circumvents Filters

Dust is an engine for generating Internet protocols used to send packets that defeat the various traffic classification techniques currently in use. There is a client and a server. In order for them to communicate, they must both be using compatible encodings. Encodings can be thought of as a generalization of the specific communication protocols used by previous network protocol obfuscation tools. Different encodings can be devised that resemble the different types of network protocol obfuscation used by previous tools. Features of the traffic can be randomized to create a “looks like nothing” protocol; alternatively, features can be shaped according to a probability distribution derived from observed traffic in order to create an encoding that mimics existing protocols.

IP and port blacklists - Like most circumvention tools in use today, a requirement is a proxy server with an IP address that has not been added to the filters blacklist. Additionally, a port must be chosen for communication that has not been blacklisted by the filter.

Packet length - Dust packets can have randomized lengths, shaped to a target distribution. Different encodings can make the packet lengths look random or like an existing protocol.

Connection headers and handshakes - Unlike typical encrypted protocols such as SSH and SSL, Dust contains no plaintext handshake. All Dust traffic is encrypted, starting with the first byte sent.

Byte sequence matching - Dust packets are encrypted, therefore the contents are randomized and recurring byte sequences are removed. In order to circumvent filters that require certain byte sequences to be present in the allowed traffic, Dust can also inject byte sequences into the encrypted traffic.

Statistical properties of content - Encrypted content can be blocked by creating a filtering rule that blocks packets with high entropy content. After encryption, Dust content is shaped to have a specified statistical distribution. This allows the high entropy encrypted content to be encoded as lower entropy, bypassing filters that flag high-entropy connections for blocking.

4.2 High-Level Protocol Overview

Dust is a polymorphic protocol engine capable of encoding and shaping network traffic to conform to specified properties. It is composed of several layers that successively transform the content to obtain filtering resistance. First, all identifying information is erased from the original traffic. Then, the results are shaped to conform to the specified protocol model. This breaks the information flow between the hidden and observable traffic, allowing for control of how the traffic is classified by the filter.

Application Layer

Every Dust communication is tied to a specific application. The first step in using Dust is to define an application-specific format for communication. In the case of using Dust as a *Pluggable Transport*, the application-specific format is simply a stream of bytes. The further layers of the protocol are agnostic as to the contents of each message and treat them as opaque byte strings. The application layer therefore has complete freedom in choosing a format. Some useful functions are provided in the Dust library for handling tasks such as cryptographic signing of messages. Any security features, such as secrecy or authentication, must be added to this layer as the other layers only provide obfuscation. In the case of Dust as a *Pluggable Transport* wrapping Tor traffic, the Tor protocol handles secrecy and authentication.

Encryption Layer

The encryption phase takes as input a plaintext message and generates an encrypted message. Unlike other encrypted protocols such as TLS, there is no plaintext header. The output of this phase is uniformly random. Only two classes of content are included in the output: encrypted data and single-use random bytes. The output of the encryption phase includes a uniformly random header followed by one or more encrypted messages. In the case of using Dust as a *Pluggable Transport*, the messages are chunks of data from the data stream. This phase in encoding is agnostic as to the contents of the plaintext message and so can be used for different applications.

The purpose of the encryption phase is to ensure a uniformly random distribution of bytes as input to the shaping phase. The encryption phase disconnects information flow from the hidden traffic to the observed traffic. It ensures that the output of the shaping phase is entirely dependent on the protocol model used and not on the original message. The encryption is only used for providing better obfuscation and is not intended to provide long-term secrecy. The encryption only needs to remain secure until the message has passed through the filter and been received at the other side. Care has nonetheless been taken to provide a secure encryption layer. However, certain features that are desirable in encryption used for long-term secrecy are not implemented when they are not appropriate for the use case. In particular, perfect forward secrecy (PFS) is not implemented. The reason is that cryptographic protocols for achieving PFS require bidirectional communication. Dust does not assume bidirectional communication and can be used to send unidirectional messages. This makes Dust more flexible in terms of the encodings that can be used as well as the underlying transports. While the current Dust implementation uses TCP for communication, Dust-encoded messages could be sent using arbitrary transports such as email messages or even using postal mail. Long-term security features such as PFS can be optionally added at the application layer when appropriate for the application. In the case of using Dust as a *Pluggable Transport*, all secrecy is provided by the Tor protocol and so this would be the appropriate place to include features such as PFS.

Shaping Layer

The shaping phase takes the uniformly random output from the encryption phase and shapes it to match the target protocol model. Each significant property of the protocol (for example, the distribution of characters in the contents or the packet length) is represented in the protocol model as a probability distribution. The encrypted byte stream is used as a psuedo-random number generator (PRNG) to sample from the probability distribution. Some properties (for example, packet timing) are not transferred losslessly and so are not used to encode information. They are sampled using a separate secure PRNG rather than the encrypted byte stream. These sampled values are then used to form the generated traffic. For example, a model that includes the properties of the packet contents, length, and timing generates a packet with a random contents, length, and timing drawn from those probability distributions. Since the sampled values are drawn from the probability distributions representing the significant properties, they conform statistically to those distributions. From the filter's point of view, the observed traffic fits the profile of the target protocol and therefore is classified as the target protocol. Since the target protocol is chosen to be one that is allowed by the filter, the encoded traffic is passed through the filter. When the traffic is received after it has passed through the filter, it is decoded to recover the original hidden traffic. Properties that were sampled using an independent PRNG (such as packet timing) are ignored. Properties that were sampled using the encrypted byte stream as a PRNG are processed by a decoder function that is the inverse of the encoder function. This reverse transformation recovers the encrypted byte stream. The encrypted byte stream is then decrypted to reveal the original plaintext message.

4.2.1 Application Layer

The first step in using Dust is to define an application-specific format for communication. In the case of using Dust as a *Pluggable Transport*, the application layer is simple. *Pluggable Transports* assume that there is one transport connection for each application connection. Therefore, the application layer opens one TCP and keeps it open until the client or server signals the end of connection. The application layer then acts as a transparent proxy. Each chunk of bytes is read from the application TCP connection, encoded, and sent over the transport TCP connection. On the other side, they are read from the transport TCP connection, decoded, and then sent over the application TCP connection.

4.2.2 Encryption Layer

The encryption layer takes as input a plaintext message generated by the application layer and generates an encrypted message. First, a key exchange must occur. Then an encrypted message can be sent. For each message, the message to be delivered is first encrypted and then a header is added specifying the necessary information to read the message.

The Key Exchange

Protocols such as SSL and SSH initiate a public key exchange using a plaintext handshake. They are therefore susceptible to protocol fingerprinting and filtering. Dust also requires a public key exchange, but does not utilize a plaintext handshake. The Dust handshake is a specific implementation of the ntor cryptographic key exchange protocol. The original ntor paper did not specify implementation details such as ciphers to be used or the wire protocol for exchanging messages. These are specified in the Dust encryption protocol specification.

To complete a key exchange, the Dust server first creates a permanent public key. The server then creates an invite, which contains the server's IP, port, public key, and the parameters for the encoding to be used. The server operator then distributes the invite to the client out-of-band. In the case of using Dust as a *Pluggable Transport* with the Tor network, the invite is distributed to Tor bridge operators and uploaded to the BridgeDB database of bridges addresses. Tor clients obtain the invite from the BridgeDB and configure the *Pluggable Transport* subsystem to use this invite, which then passes it to the Dust client.

The client then generates an ephemeral public key for use only on the current connection and then uses the IP and port information from the invite to connect to the server. The client and server then exchange ephemeral public keys and the client provides proof that it knows the server's permanent public key. This provides secrecy for the encryption key as well as defends against active attacks that probe random IP addresses looking for servers. The encryption key is derived from the ephemeral public keys. At this point, the client generates a random initialization vector and sends it to the server. This IV along with the encryption key allows for a conversation consisting of one or more encrypted messages to begin. In the case of using Dust as a *Pluggable Transport*, the messages contain chunks of data from the data stream.

Sending Messages

Once the conversation has begun, messages can be sent. To send a message, the encryption key and the initialization vector are used to encrypt first the message length and then the message. This creates an entirely encrypted, random-looking sequence of bytes. This completes the encryption layer encoding of the message. The message is then sent onto the shaping layer to be encoded according to the probability distributions of the model.

When the server receives an encrypted message that has been decoded by the shaping layer, it decrypts the message length and uses it to extract a message of the correct length. As bytes may be added to the end of the message to adjust its length to the target protocol parameters, the length is necessary in order to know which data to keep and which to discard. Once the message is decrypted, it is delivered to whatever backend service the server is providing. In the case of using Dust as a *Pluggable Transport*, the decrypted message contains a chunk of data that is added to the data stream.

Randomness

A key consideration in the Dust protocol is that the only information in an encrypted Dust message (before shaping) should effectively always be uniformly random to an observer. There are therefore only two types of information that are included in an encrypted Dust packet: encrypted bytes, which should appear random, and single-use random bytes that are generated by a PRNG. The apparent randomness is essential to the protocol's ability to avoid detection. Therefore, care should be taken to use a good random number generator and to never reuse random bytes. For instance, the client's ephemeral public key is intended for a single use and should never be reused. The server's permanent public key is intended for multiple uses and so should never be sent over the wire. Similarly, the initialization vector used in data packets should always be randomly generated and should never be reused between messages.

Wire Protocol

The cryptographic protocol and wire protocol used to establish secure communication using the uniformly random byte stream is fully documented in the protocol specification section. For the purpose of understanding the shaping layer section, the important consideration for the encryption layer is that the application layer data goes into the encryption layer and results in a uniformly random encrypted byte stream for use in the shaping later.

4.2.3 Shaping Layer

The shaping layer takes the uniformly random output from the encryption layer and shapes it to match the target statistical model for each feature. Each statistical property of the traffic is represented in the statistical model as a probability distribution. The encrypted byte stream is used as a psuedo-random number generator (PRNG) to sample from the probability distribution. Some properties (for example, packet timing) are not transferred losslessly and so are not used to encode information. They are sampled using a separate secure PRNG rather than the encrypted byte stream. These sampled values are then used to form the generated traffic.

The shaping later determines what traffic is actually sent over the network. It takes as input the encrypted stream from the encryption layer. However, traffic sent over the network is determined independently of whether or not there is actual data to send. This is achieved by decoupling in the API. There is one API call to register new data to be sent, and another API call to obtain the actual bytes that are sent over the network. The latter always succeeds, regardless of whether the former has been called. Therefore, when there are bytes to send they are used, and when there are no bytes to send random padding is used.

Shaping Features

As long as there is a transport connection open between the client and the server, Dust continues to send packets, whether or not there is any actual data to send. In order to determine how many packets to send, a number is obtained from the Dust engine using the flow model. Unlike some other protocol obfuscation tools, Dust does not model packet inter-arrival times, but instead models the flow rate in terms of the number of packets per second. Each time the Dust engine is queried for the number of packets to send, the number of milliseconds elapsed since the last set of packets were sent is supplied.

Once a number of packets to send has been determined, the length of each packet is determined next. The Dust engine is queried for a length for each packet, using the packet length model. For each packet to be sent, bytes equal in number to the packet's length are obtained from the engine. These bytes may or may not contain encrypted data. If they do not contain encrypted data, then they contain random padding. Once a sequence of bytes has been obtained by combining encrypted data with optional random padding, this sequence of uniformly random bytes is encoded using reverse Huffman encoding and the probability distribution supplied by the content model. The result is a lower entropy byte sequence with a content distribution approximating that of the content model.

To decode a received message, packet timing and length is ignored as these characteristics may be altered during transmission. The bytes of the packet content are decoded and decrypted. Bytes that contain padding are discarded. The remaining bytes contain an encrypted message that is passed to the encryption layer for decryption and then on to the application layer.

An important thing to note about the client/server communication process with Dust is that the traffic sent over the network always follows the specified target distribution for each property regardless of any other concerns. Servers continue to communicate with clients even when decoding and decryption of the received messages fails. The reason behind this is that the traffic sent over the network should never reveal any information about the underlying messages being sent.

4.3 Protocol Specifications

4.3.1 Application Layer Protocol Specification

Overview

The application protocol allows for proxied connection byte streams to be transmitted over transport connections. The use case for this protocol is when Dust is used as a transparent proxy, as in when it is used as a *Pluggable Transport*.

Protocol

The client begins by either initiating a new session or continuing an existing session. When creating a new session, the server returns the session ID for the new session. A number of data messages are then sent that include the session ID and the sequence number starting at 0. At any time, either side can send a close message to end the session. For particularly long-lived sessions, the sequence numbers may be insufficient for the number of messages to be sent. In this case, an "Extend Session Request" message can be sent to reset the sequence number to zero and get a new session ID which extends the current session.

Starting a new session

- Client request
    - 1 byte - command code, 0x00 New Session Request
- Server response
    - 1 byte - command code, 0x01 New Session Okay
    - 32 bytes - new session ID

Continuing an existing session

- Client request
  - 1 byte - command code, 0x02 Continue Session Request
  - 32 bytes - session ID to continue
- Server response
  - 1 byte - command code, 0x03 Continue Session Okay
  - 32 bytes - session ID to continue

Extending a session

- Client request
  - 1 byte - command code, 0x04 Extend Session Request
  - 32 bytes - session ID to extend
- Server response
  - 1 byte - command code, 0x05 Extend Session Okay
  - 32 bytes - new session ID

Closing a session

- Request (can be client or server)
  - 1 byte - command code, 0x06 Close Session Request
  - 32 bytes - session ID to close
- Response (other side, server or client)
  - 1 byte - command code, 0x07 Close Session Okay
  - 32 bytes - session ID to close

Sending data

- 1 byte - command code, 0x0F Data
- 32 bytes - session ID
- 2 bytes - sequence number
- 2 bytes - length
- Variable – data

Errors

- 1 byte - error code, 0xF0 Too Many Sessions
- 1 byte - error code, 0xF1 Unknown Session ID
- 1 byte - error code, 0xF2 Connection Is Closed
- 1 byte - error code, 0xF3 Bad Sequence Number

4.3.2 Encryption Layer Protocol

Introduction

The encryption protocol consists of a secure key exchange protocol with server identity verification and a wire protocol for transmitting the key exchange and messages encrypted with the exchanged key.

Handshake Protocol

The handshake protocol is based on ntor. It has been modified from the original ntor paper in order to be compatible with the needs of obfuscation. Each step of the ntor protocol is discussed in the next section. In the following section, there is discussion of which steps were changed for use in Dust and the security implications of these changes.

The ntor Protocol, Step by Step

When $\underline{B}$ is initialized as a server:

1. Set b $\leftarrow^{\$}$ {1, . . . , q − 1} and set B ← g^b.
2. Set (b, B) as $\underline{B}$'s static key pair.
3. Set $cert_B$ = ($\underline{B}$, B) as $\underline{B}$'s certificate.

When initializing the server, no messages are output. The server generates a key pair [1,2] and the public key becomes the server's certificate [3].

When $\underline{A}$ is initialized as a client:

4. Obtain an authentic copy of $\underline{B}$'s certificate.

When initializing the client, the server's certificate is delivered out of band [4], along with the other information necessary for contacting the server such as the server's IP address and port.

When $\underline{A}$ receives the message (params, pid) = ((“new session”, ntor), $\underline{B}$):

5. Verify that $\underline{A}$ holds an authenticated certificate $cert_B = (\underline{B}$,B).
6. Obtain an unused ephemeral key pair (x, X ← g^x); set session id $\Psi_a = H_{sid}(X)$.
7. Set $M^{\underline{A}}_{state}[\Psi_a] \leftarrow (ntor, \underline{B}, x, X)$.
8. Return session identifier Ψ_a and outgoing message $msg' \leftarrow (ntor, \underline{B}, X)$.

The “new session” message is conceptual and not actually output. It represents the event in which the client seeks to initiate a new session with the server.

In order for the client to connect to the server, the client and the server both must already be initialized [1,2,3] and the server's certificate must have been transmitted out of band to the client [4,5].

The client generates an ephemeral key pair that is only used for this session [6]. A hash of the public key of the client's ephemeral keypair is used as the session identifier [6]. The client stores session information in memory consisting of the ephemeral key pair and the server's public key, using the session identifier as the key for retrieving the session information [7].

The output is a message generated by concatenating the static string "ntor", the server's public key, and the client's ephemeral public key [8]. This message requests from the server that a new session be established.

When $\underline{B}$ receives the message msg = (ntor, $\underline{B}$, X):

9. Verify X ∈ G^*.

10. Obtain an unused ephemeral key pair (y, Y ← g^y); set session id $\Psi_b \leftarrow H_{sid}(X)$.

11. Compute $(sk', sk) = H(X^y, X^b, \underline{B}, X, Y, ntor)$.

12. Compute $t_B = H_{mac}(sk', \underline{B}, Y, X, ntor, "server")$.

13. Return session identifier Ψ_b and outgoing message $msg' \leftarrow (ntor, Y, t_B)$.

14. Complete Ψ_b by deleting y and outputting $(sk, *, (\underline{v_o}, \underline{v_1}))$, where $\underline{v_o} = (X)$ and $\underline{v_1} = (Y, B)$.

When the server receives a message requesting a new session be established, it first checks that the client's ephemeral public key is a valid public key [9].

The server then generates an ephemeral key pair that is only used for this session [10]. A hash of the public key of the server's ephemeral keypair is used as the session identifier [10].

The server then generates a shared key and a confirmation code. The shared key is calculated by taking a hash of the following values: the key generated by an ECDH between the client's ephemeral public key and the server's ephemeral private key, the key generated by an ECDH between the client's ephemeral public key and the server's static private key, the client's ephemeral public key, the server's ephemeral public key, and the string "ntor" [11]. The shared key is then used to calculate the confirmation code. The confirmation code is calculated by taking an HMAC of the following values: the shared key, the server's identifier (for instance, the server's IP address), the server's public key, the client's public key, the string "ntor", and the string "server" [12].

The server responds to the client's request for a new session with a message consisting of the concatenation of the following values: the string "ntor", the server's ephemeral public key, and the confirmation code [13].

Once the confirmation code has been sent, the server forgets its ephemeral private key (this is what makes it ephemeral) and records the session information in association with the session identifier. The session information consists of the following values: the shared key, the client's ephemeral public key, the server's ephemeral public key, and the server's static public key.

When $\underline{A}$ receives the message $msg' \leftarrow (ntor, Y, t_B)$ for session identifier Ψ_a:

15. Verify session state $M^{A}_{state}[\Psi a\ ]$ exists.
16. Retrieve $\underline{B}$, x, and X from $M^{A}_{state}[\Psi a\ ]$.
17. Verify Y $\in G^*$.
18. Compute $(sk', sk) = H(Y^x, B^x, \underline{B}, X, Y, ntor)$.
19. Verify $t_B = H_{mac}(sk', \underline{B}, Y, X, ntor, "server")$.
20. Complete Ψ_a by deleting $M^{A}_{state}[\Psi a\ ]$ and outputting $(sk, \underline{B}, (\underline{v_o}, \underline{v_1}))$, where $\underline{v_o} = (X)$ and $\underline{v_1} = (Y, B)$.

When the client receives a response from the server, it first checks to see if it did in fact request a session with that server [15]. If so, it retrieves the session information, which contains the server's identifier (for instance, the server's IP address), the client's ephemeral private key, and the client's ephemeral public key [16].

The client next verifies that the server's ephemeral public key is a valid public key [17]. The client then generates a shared key and a confirmation code. The shared key is calculated by taking a hash of the following values: the key generated by an ECDH between the server's ephemeral public key and the client's ephemeral private key, the key generated by an ECDH between the client's ephemeral private key and the server's static public key, the client's ephemeral public key, the server's ephemeral public key, and the string "ntor" [18]. The shared key should be identical to the one calculated on the server. The shared key is then used to calculate the confirmation code. The confirmation code is calculated by taking an HMAC of the following values: the shared key, the server's identifier (for instance, the server's IP address), the server's public key, the client's public key, the string "ntor", and the string "server" [19]. The confirmation code calculation is identical to the one used on the server and should have an identical result if the shared key is, at it should be, also identical between the client and server.

The client then deletes the session state, including its ephemeral private key, and stores a new session state consisting of the shared key, the server identifier, the client's ephemeral public key, the server's ephemeral public key, and the server's static public key.

Once this last step has been completed, the client and server both have the same shared key and they have verified by virtue of the confirmation code that the other side also has the correct shared key. At this point, encrypted communication can occur using the shared key as the encryption key.

Problems with ntor

There are two messages that are exchanged between the client and server in the ntor protocol. The out-of-band exchange of the server's static public key is not counted here. The first message is sent from the client to the server requesting a new session and the second message is sent from the server to the client in response. The client's message contains the following: the string "ntor", the server's identifier, and the client's ephemeral public key. The server's message contains the following: the string "ntor, the server's ephemeral public key, and a confirmation code.

In order for a handshake to work with obfuscating protocols, all output messages must be free of information that can identify it as belonging to an obfuscating protocol. If ntor is to be used in obfuscating protocols, then identifying characteristics such as transmitting the string "ntor" at the beginning of the handshake are undesirable.

Specifically, the requirements set forth by the Dust project for wire protocols, including handshake protocols, are that all bytes contained in output messages must be either encrypted or random single-use values (nonces). Given this criteria, we can look at the ntor messages and determine what changes are necessary to the protocol in order for it to be acceptable.

The client's message contains the following: the string "ntor", the server's identifier, and the client's ephemeral public key. The string "ntor" is neither encrypted nor random and must be removed from the protocol. The server's identifier could be something non-random like an IP, or it could be a randomly generated string. However, since the same server identifier is used for each new session initiated with the same server, it is not single-use. Therefore, it does not meet the requirements and must be removed. The client's ephemeral public key is random because it is an Elligator public key, and they are guaranteed to be uniformly random. Also, because it is ephemeral it is single use. Therefore, it meets the requirements and can remain. The confirmation code is a special case. It is the result of a hash function and if a cryptographic hash function is used then the result should be uniformly random bytes. However, if this hash can be calculated by an observer, then it can be used to detect the presence of Dust traffic and so could not be allowed. Examining how the hash function is calculated, two of the inputs are the results of ECDH calculations. These calculations require private keys that are never transmitted. Ephemeral keys are also used, which are single use. Therefore, the confirmation code is a single-use value since the ephemeral keys used to generate it is not used again. Since it cannot be calculated by an observer, it can be considered a random single-use value and is acceptable.

The modified version of the ntor protocol that fits the requirements therefore consists of a client message containing just the client's ephemeral public key, followed by a server response containing the server's ephemeral public key and the confirmation code. One can assume that the parts removed from the ntor protocol were added for a reason and that removing them could have some impact on the security of the modified ntor protocol. While the security arguments made in the ntor paper do not reference these parts of the protocol, the authors may have had some other attacks in mind that were not documented in the paper. Fortunately, these changes to the Dust implementation of ntor were discussed with one of ntor's authors, Ian Goldberg, and it was decided that they do not break the security of ntor. There is, however, one item of concern, which is that a server identity value (which is passed out of band) is used in some of the calculations for ntor. However, there are attacks that are possible if the server identity is not verified. In the use case in which Dust is used as a *Pluggable Transport*, the server identity is passed in from the *Pluggable Transport* subsystem and Dust has no way of verifying it. It is thus the responsibility of the *Pluggable Transport* subsystem to verify the validity of the server identity. Unfortunately, currently no such verification is done. This represents a vulnerability for all *Pluggable Transport*s that use variants of ntor requiring a verified server identity. While outside the scope of this project to fix, this is something that should be addressed in the Tor architecture in order to prevent known attacks.

Dust Handshake

Initializing a server

- The server generates a static ECDH keypair using Curve25519 that becomes the server's certificate.

Initializing a client

- The client obtains a copy of the server's certificate out of band, along with the server's IP and port

Creating a new session

- Verify that the client has the certificate for the server
- Create an ephemeral Curve25519 keypair, associating it with the given server
- Encrypt the ephemeral public key with Elligator
- Send the encrypted ephemeral public key to the server

When the server receives a client request for a new session

- Obtain the client's encrypted ephemeral public key from the message.
- Decrypt the client's encrypted ephemeral public key with Elligator to get the client's Curve25519 ephemeral public key
- Create an ephemeral Curve25519 keypair
- Create a shared key (see below)
- Create a server confirmation code using the shared key (see below)
- Encrypt the ephemeral public key with Elligator
- Send the encrypted ephemeral public key and server confirmation code to the client
- Delete the ephemeral private key
- Store the shared key, associated with the client

When the client receives a server response for a new session

- Obtain the server's encrypted ephemeral public key and the server confirmation code from the message.
- Decrypt the server's encrypted ephemeral public key with Elligator to get the server's ephemeral Curve25519 public key
- Create a shared key (see below)
- Create a client confirmation code using the shared key (see below)
- Verify that the client confirmation code and server confirmation code are identical
- Delete the ephemeral private key
- Store the shared key, associated with the server

Creating a shared key

- On the server, hash with Skein-256-256 the following:
    - 32 bytes - ECDH(server's ephemeral private key, client's ephemeral public key)
    - 32 bytes - ECDH(server's static private key, client's ephemeral public key)
    - 7 or 19 bytes - server identifier (see below)
    - 32 bytes - client's ephemeral public key
    - 32 bytes - server's ephemeral public key
    - 4 bytes - "ntor"
- On the client, hash with Skein-256-256 the following:
    - 32 bytes - ECDH(client's ephemeral private key, server's ephemeral public key)
    - 32 bytes - ECDH(client's ephemeral private key, server's static public key)
    - 7 or 10 bytes - server identifier (see below)
    - 32 bytes - client's ephemeral public key
    - 32 bytes - server's ephemeral public key
    - 4 bytes - "ntor"

Creating a confirmation code

- Identical on both client and server, HMAC with Skein-256-256 and the shared key the following:
    - 7 or 19 bytes - server identifier (see below)
    - 32 bytes - server's ephemeral public key
    - 32 bytes - client's ephemeral public key
    - 4 bytes - "ntor"
    - 6 bytes - "server"

Creating a server identifier

- For an IPv4 address:
    - 1 byte - flags (see below)
    - 4 bytes - IP Address
    - 2 bytes - port
- For an IPv6 address:
    - 1 byte - flags (see below)
    - 16 bytes - IP Address
    - 2 bytes - port

Creating server identifier flags

- 0 - 0 for IPv4, 1 for IPv6
- 1 - 0 for TCP, 1 for UDP
- 2-7 - Reserved, set to 0

Wire Protocol

The following reduces the complexity of the handshake protocol description into just what is actually output:

- Client Request to Server
    - 32 bytes - client's ephemeral public key
- Server Response to Client
    - 32 bytes - server's ephemeral public key
    - 32 bytes - confirmation code

Message Protocol

The final result of the handshake protocol is that the client and server both have an identical shared key, which can be used for encrypting messages. Once the handshake is complete, message transmission can commence. For this phase of communication, the encryption protocol provides semantics for message transmission.

The message transmission protocol begins with a header that consists of a random initialization vector (IV). After the header, any number of records can be sent until such point as the session is ended. The record format consists of an encrypted length followed by an encrypted payload of the specified length. The payload contains a header, data, and a verification code. As the client and server use the same shared key, the message protocol is identical for both client and server.

Message Protocol Step-by-step

Sending a message with data

- If this is the first message, send the header (see below)
- Generate flags (see below)
- Generate a verification code for the data (see below)
- Create a payload by concatenating the following:
    - 1 byte - flags
    - Variable - data
    - 32 bytes - verification code
- Encrypt the payload with the shared key to get the encrypted payload
- Obtain the length of the payload and encode it as a 2-byte value in network byte order
- Encrypt the length with the shared key to get the encrypted length
- Create a record by concatenating the following:
    - 2 bytes - encrypted length
    - Variable - encrypted payload
- Deliver the message

Sending a message without data

- If this is the first message, send the header (see below)
- Generate flags (see below)
- Generate randomly sized blank data
- A random number of 0-valued bytes
- Generate a blank verification code for the data
    - 32 0-valued bytes
- Create a payload by concatenating the following:
    - 1 byte - flags
    - Variable - blank data
    - 32 bytes - blank verification code
- Encrypt the payload with the shared key to get the encrypted payload
- Obtain the length of the payload and encode it as a 2-byte value in network byte order
- Encrypt the length with the shared key to get the encrypted length
- Create a record by concatenating the following:
    - 2 bytes - encrypted length
    - Variable - encrypted payload
- Deliver the message

Sending the header

- Generate a random 32-byte initialization vector (IV)
- Deliver the IV

Generate flags

- 0 - 0 for no data, 1 for data included
- 1-7 - Reserved, set to 0

Generate a verification code for the data

- HMAC with the shared key the following:
    - 2 bytes - unencrypted length
    - 1 byte - flags
    - Variable - data

Wire Protocol

The following reduces the complexity of the message protocol description into just what messages are actually output. The client and server side use identical wire protocols.

- 32 bytes - IV
- Repeating
    - 2 bytes - encrypted length
    - Variable - encrypted payload
    - If data
        - 1 byte - flags
        - Variable - data
        - 32 bytes - verification code
    - otherwise
        - 1 byte - flags
        - Variable - blank data
        - 32 bytes - blank verification code

4.4 CONCLUSION

The Dust engine was implemented using three layers: the application layer, the encryption layer, and the shaping later. The application layer for Dust as a *Pluggable Transport* is a single set of frames including chunks of a byte stream to be forwarded. The encryption layer uses a variant of the ntor protocol to establish an encrypted communication channel that is secure against passive observers and active probing attacks. The shaping layer encodes uniformly random encrypted data to follow a target probability distribution provided by the model.

5. Evaluation

5.1 Evaluation Methodology

The evaluation of the overall research project depended on the results of evaluating each phase of research. The evaluation answered the following questions to determine the success of the project:

- How accurate are the models of filters?
- How effective and efficient is the circumvention engine?
- How effective and efficient are the models?

The models were evaluated using 2-fold cross-validation of goodness-of-fit. This determined the goodness of fit of the models to the data used to build them. The effectiveness of the circumvention engine was tested against a simulated adversary constructed by using the filter models as binary classifiers. Normal traffic as well as traffic generated by the engine was classified into blocked or allowed categories by the simulated adversary. The accuracy of the simulated adversary was used to determine how accurate the filter is at classifying normal traffic as well as encoded traffic. Efficiency of the circumvention engine was measured by calculating the overhead of Dust encodings as determined by the size of the original message compared to the amount of actual network traffic sent. Together these evaluations answer the overall research question of the project, which is how well the proposed method of circumventing can restore access to credible and relevant information that was lost due to Internet filtering.

The goal of evaluation is to measure the effectiveness and efficiency of the Dust engine. To evaluate effectiveness, the engine was tested against simulated adversaries. The chosen protocols for encoding using Dust were HTTP, HTTPS, and RTSP. Therefore, adversaries were created to differentiate traffic using these protocols. Six adversaries were created using traffic for these protocols. First, three adversaries were created as baselines. The baseline adversaries were HTTP-HTTP, HTTPS-HTTPS, and RTSP-RTSP. With the baseline adversaries, the same protocol was used for both categories. However, the same data was not always used for both categories. The datasets were once again divided into two sets, which we can call X and Y. Different combinations of data were used for training and testing. For instance, in one variant the HTTP X dataset was used for training one side and the HTTP Y dataset was used for training the other side. In other variations, the same dataset was used for both sides. The purpose of this set of adversaries is to establish a baseline for accuracy of the classifier when no interesting signal exists. The accuracy of the baseline adversaries is discussed below. The second set of three adversaries differentiated between the combinations of the three different protocols. The adversaries are as follows: HTTP-HTTPS, HTTPS-RTSP, and HTTP-RTSP. These adversaries were similarly built in several variations using combinations of data from the X and Y subset for each protocol. The accuracy of the second set of adversaries is also discussed below. Additionally, these three adversaries were also used to evaluate Dust. The same adversaries were used to classify traffic, but one set of data used during testing was substituted for Dust encoded data. For instance, the HTTP-HTTPS adversary was used to classify a mixture of DustHTTP traffic and HTTPS traffic. This evaluation is discussed in detail in the evaluation section. Finally, a new set of adversaries was created specifically to classify Dust-encoded traffic. Three adversaries were created of this type: DustHTTP-HTTP, DustHTTPS-HTTPS, and DustRTSP-RTSP. The purpose in creating these adversaries was to see if new adversaries trained on Dust traffic could be created that could differentiate between Dust-encoded traffic and the traffic it was attempting to mimic. All of the different tests for these adversaries can be seen in Appendix C, and the results are summarized below.

The encodings were evaluated by looking at the performance of the adversaries with and without encoding. The metrics used to evaluate were goodness of fit and overall accuracy of prediction. Root Mean Squared Error (RMSE) was used as the goodness-of-fit metric. Accuracy was calculated by dividing the number of true positives and true negatives by the total number of predictions. So false positives and false negatives counted against accuracy, as did connections that the adversary was not able to classify. A successful encoding is one that reduces the filter's precision when moving from normal traffic to encoded traffic.

Efficiency was measured by calculating the overhead of the Dust encodings. The overhead was determined by the size of the original message compared to the amount of actual network traffic sent. There is always some overhead when obfuscating traffic. Many obfuscated protocols have a fixed amount of overhead. For instance, protocols such as obfs2 [20] have a header that must be sent before any data. After the header, data is merely encrypted and so there is no additional overhead. Other protocols have variable, but usually low, overhead. For instance, FTE is reported to have "as little as 16% bandwidth overhead compared to standard SSH tunnels" [8]. Dust is different in that the original traffic and encoded traffic are disconnected. If the target protocol has a bandwidth usage of 1 Mbps then the encode traffic uses 1 Mbps regardless of the bandwidth of the original traffic. In fact, a Dust conversation can be initiated that has no source traffic whatsoever and uses a PRNG to generate traffic. This configuration is used in some of the testing tools developed for examining the characteristics of filtering hardware. While the worst-case scenario for Dust is therefore unbounded overhead, the average case is better. In all tests, the bandwidth was adequate to perform the tests over a consumer grade home Internet connection.

5.2 Results

Figures 40 through 47 show the goodness of fit for the simulated adversaries. There is a chart for each feature for both incoming and outgoing directions. For each protocol, the data was divided into two sets X and Y. Each of these two sets was used to train a different protocol model and then tested against each of the two sets. On the x-axis, for each protocol results first for the XX (train using X and then test against X) and YY trials are shown, followed by the XY and YX trials. On the left side are the original protocols observed in the field: HTTP, HTTPS, and RTSP. On the right side of the graph are the dust encodings seeking to mimic those protocols: DustHTTP, DustHTTPS, and DustRTSP. The y-axis shows the root mean squared error (RMSE) between the predictions of the model and the test data.

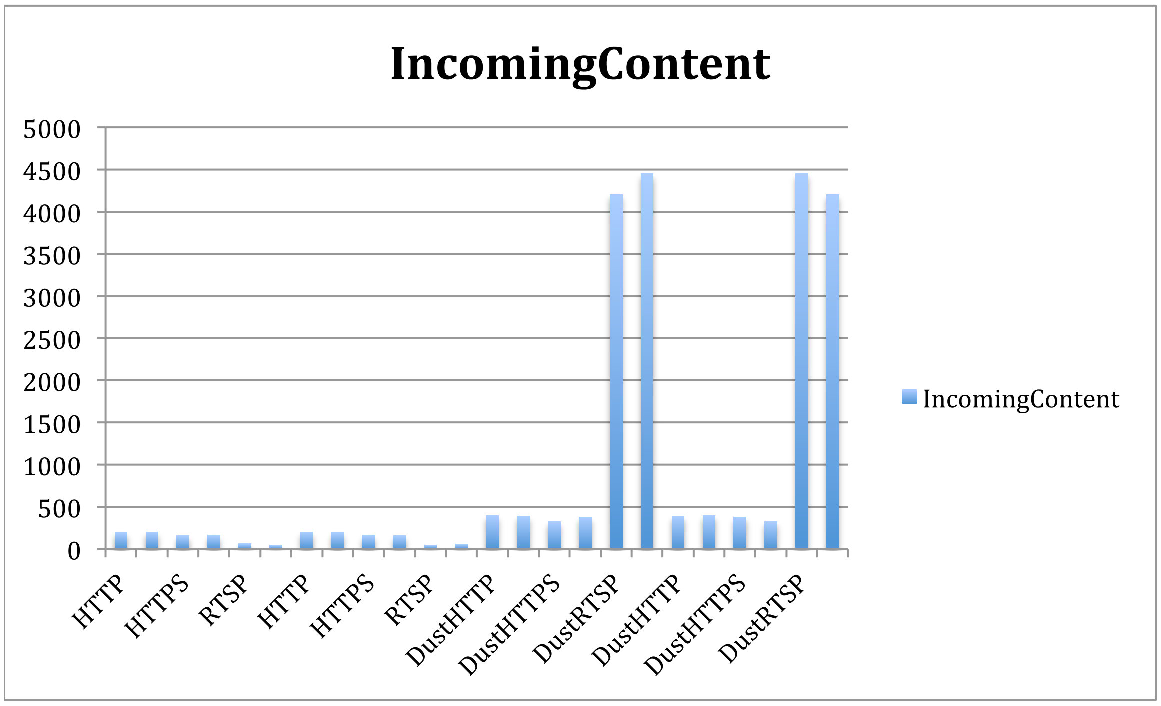

Figure 40: RMSE of predictive model for incoming content

For the incoming content model shown in Figure 40, RMSE scores show a somewhat better fit for HTTP and HTTPS than for DustHTTP and DustHTTPS. RTSP has an especially good fit and DustRTSP has an especially bad fit. The results are consistent across XX, YY, XY, and YX trials.

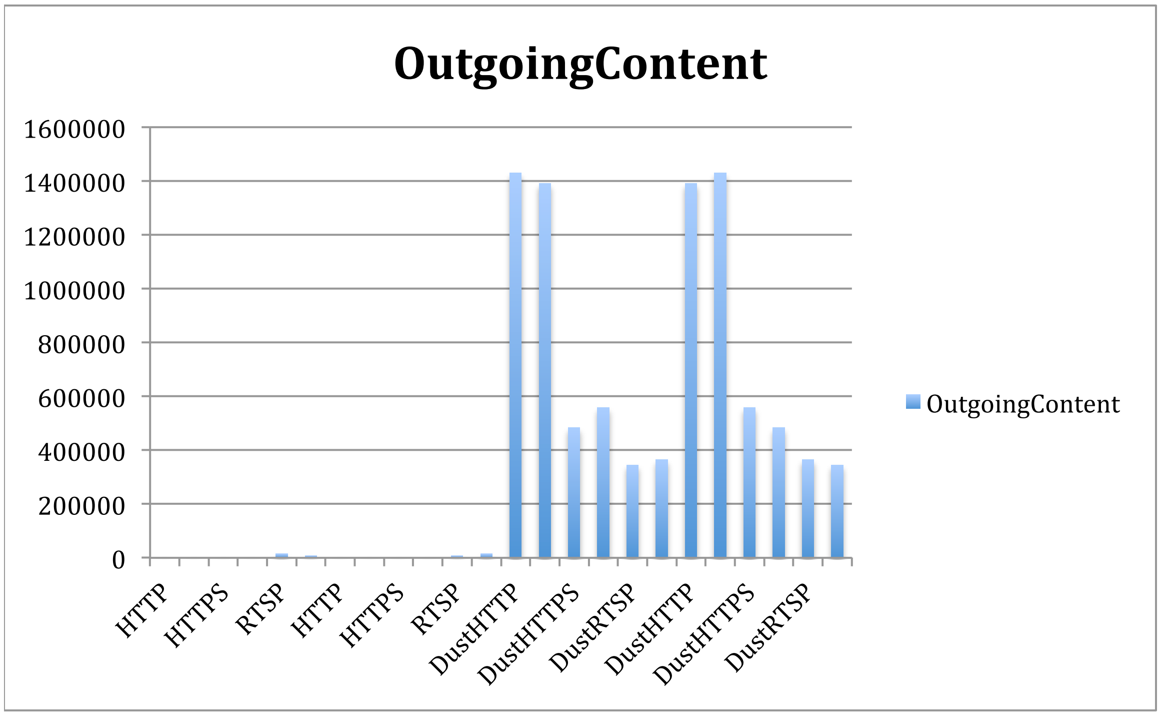

Figure 41: RMSE of predictive model for outgoing content

For the outgoing content model shown in Figure 41, RMSE scores for HTTP, HTTPS, and RTSP are so much better than for DustHTTP, DustHTTPS, and DustRTSP that they barely show up on the graph. The DustHTTP model has an order of magnitude worse fit than the DustHTTPS and DustRTSP models. The results are consistent across XX, YY, XY, and YX trials.

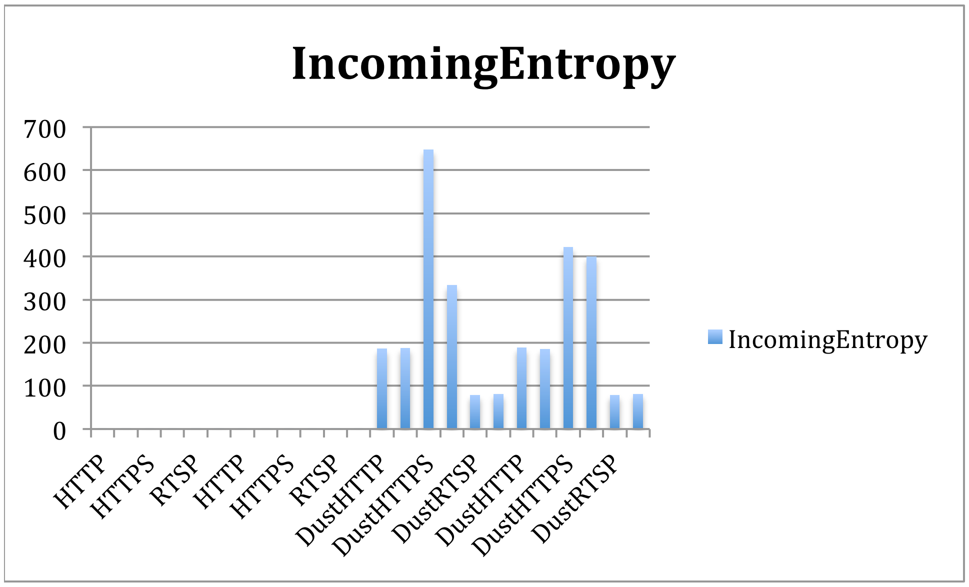

Figure 42: RMSE of predictive model for incoming entropy

For the incoming entropy model shown in Figure 42, RMSE scores for HTTP, HTTPS, and RTSP are two orders of magnitude better than for DustHTTP, DustHTTPS, and DustRTSP. The DustHTTPS model has a significantly worse fit than the DustHTTP and DustRTSP models. The results are mostly consistent across XX, YY, XY, and YX trials, which the exception of DustHTTPS. This model shows sensitivity to the training and testing sets. It does much worse when both trained and tested with set X and much better when both trained and tested with set Y. The XY and YX trails average out the difference and show consistent results.

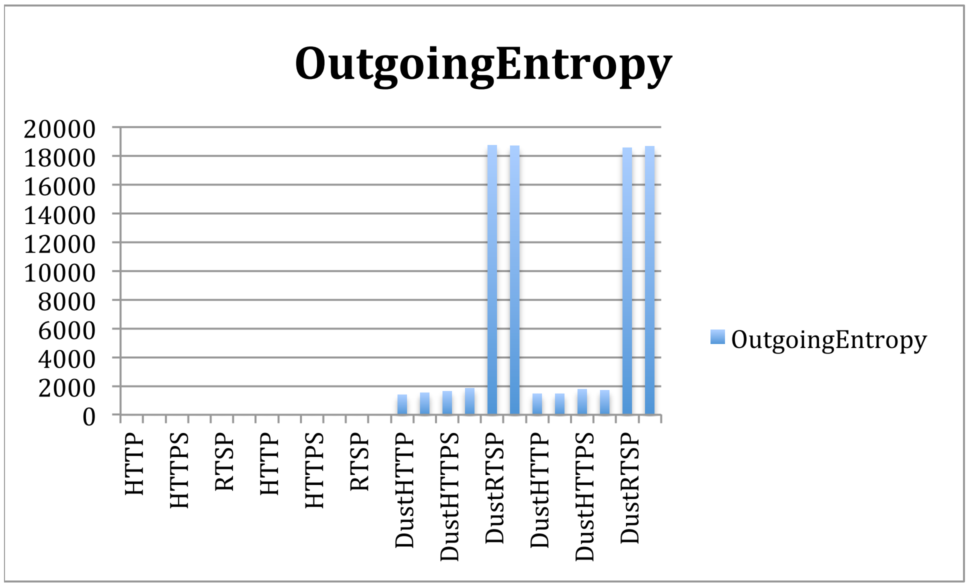

Figure 43: RMSE of predictive model for outgoing entropy

For the outgoing entropy model shown in Figure 43, RMSE scores for HTTP, HTTPS, and RTSP are three orders of magnitude better than for DustHTTP and DustHTTPS, and five orders of magnitude better than DustRTSP. The results are consistent across XX, YY, XY, and YX trials.

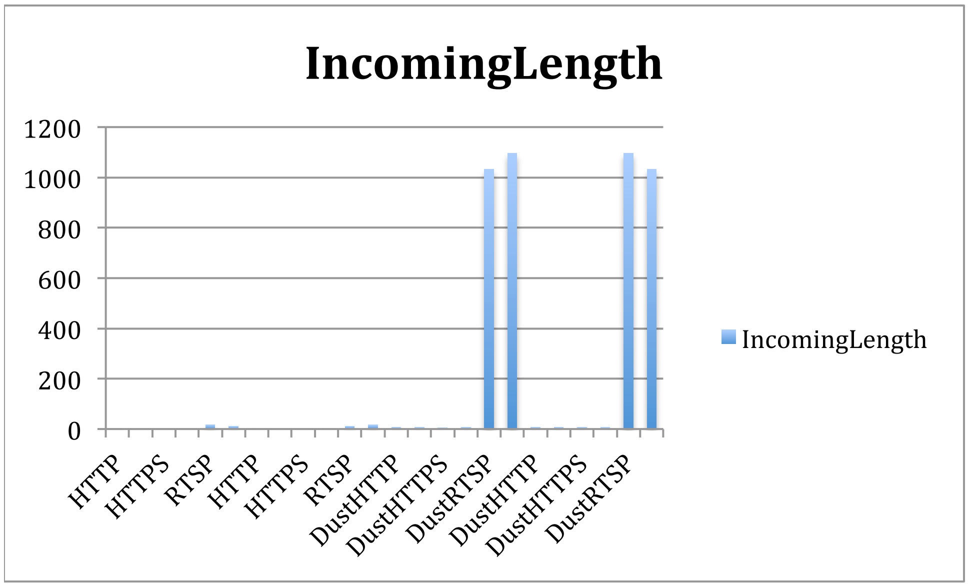

Figure 44: RMSE of predictive model for incoming packet length

For the incoming packet length model shown in Figure 44, HTTP, HTTPS, RTSP, DustHTTP, and DustHTTPS all show good fits. DustRTSP shows a three orders of magnitude worse fit. The results are consistent across XX, YY, XY, and YX trials.

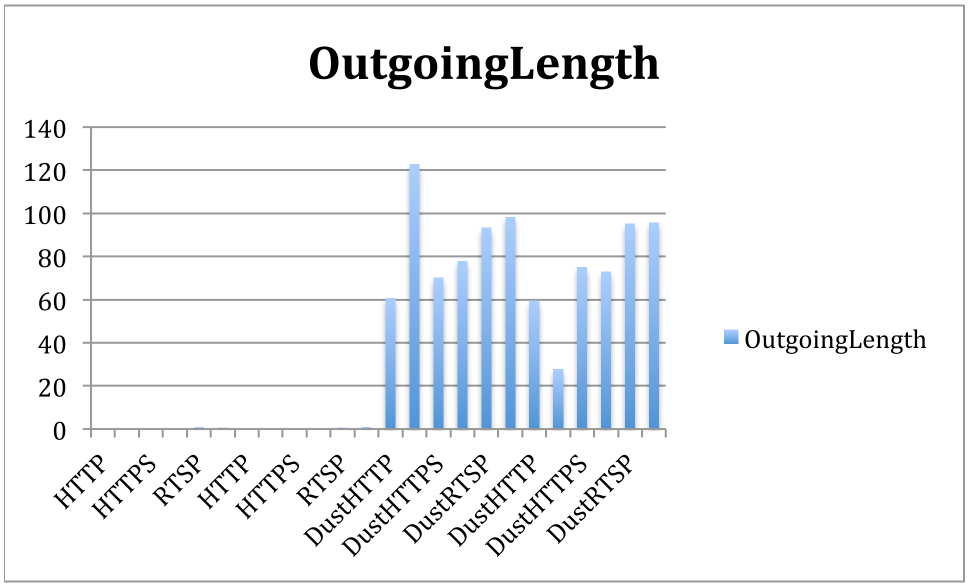

Figure 45: RMSE of predictive model for outgoing packet length

For the outgoing packet-length model shown in Figure 45, HTTP, HTTPS, and RTSP show an exceedingly good fit. DustHTTP, DustHTTPS, and DustRTSP show one to two orders of magnitude worse fit. The results vary based on the training and testing datasets. For these three protocols, they show consistently better fits when trained on set X and tested on set X than when trained and testing on set Y. However, DustHTTPS and DustRTSP show consistent results when different datasets are used for training and testing. Only DustHTTP shows inconsistent results when using different training and testing datasets, with three times better fit on the YX trail than on the XY trail.

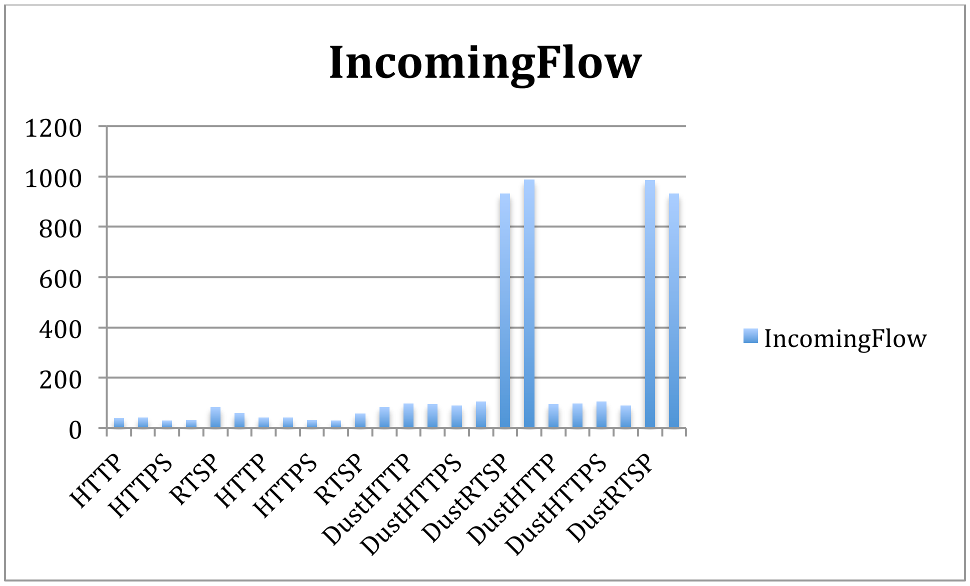

Figure 46: RMSE of predictive model for incoming packet flow

For the incoming packet flow model shown in Figure 46, HTTP, HTTPS, RTSP, DustHTTP, and DustHTTPS all show good fits. DustRTSP shows a two orders of magnitude worse fit. The results are consistent across XX, YY, XY, and YX trials.

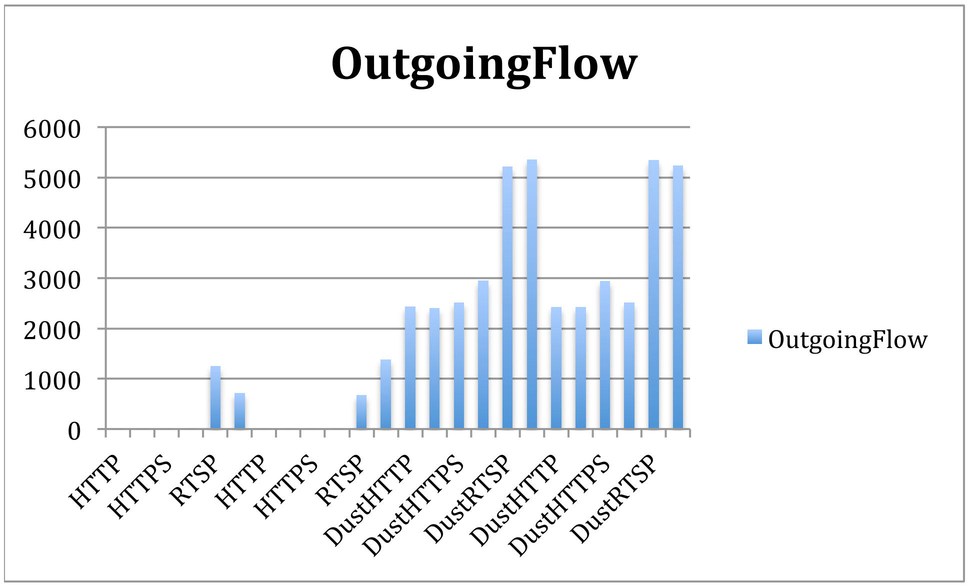

Figure 47: RMSE of predictive model for outgoing packet flow

For the outgoing packet flow model shown in Figure 47, HTTP and HTTPS show good fits. RTSP, DustHTTP, DustHTTPS, and DustRTSP show a two orders of magnitude worse fit. Of these, RTSP is the best fit and DustRTSP is the worst fit. The results are mostly consistent across XX, YY, XY, and YX trials. RTSP shows a sensitivity to the training and testing datasets, with the error doubling for the YY and XY trails compared to the XX and YX trails.

Overall, the error varies widely between features, directions, and protocols. However, it generally does not vary much when switching between the dataset used for training and the dataset used for testing. The goodness-of-fit testing therefore shows that the performance of the model is not sensitively dependent on the data set used to train it for most features and protocols. Figures 48 and 49 explore changes in error more closely.

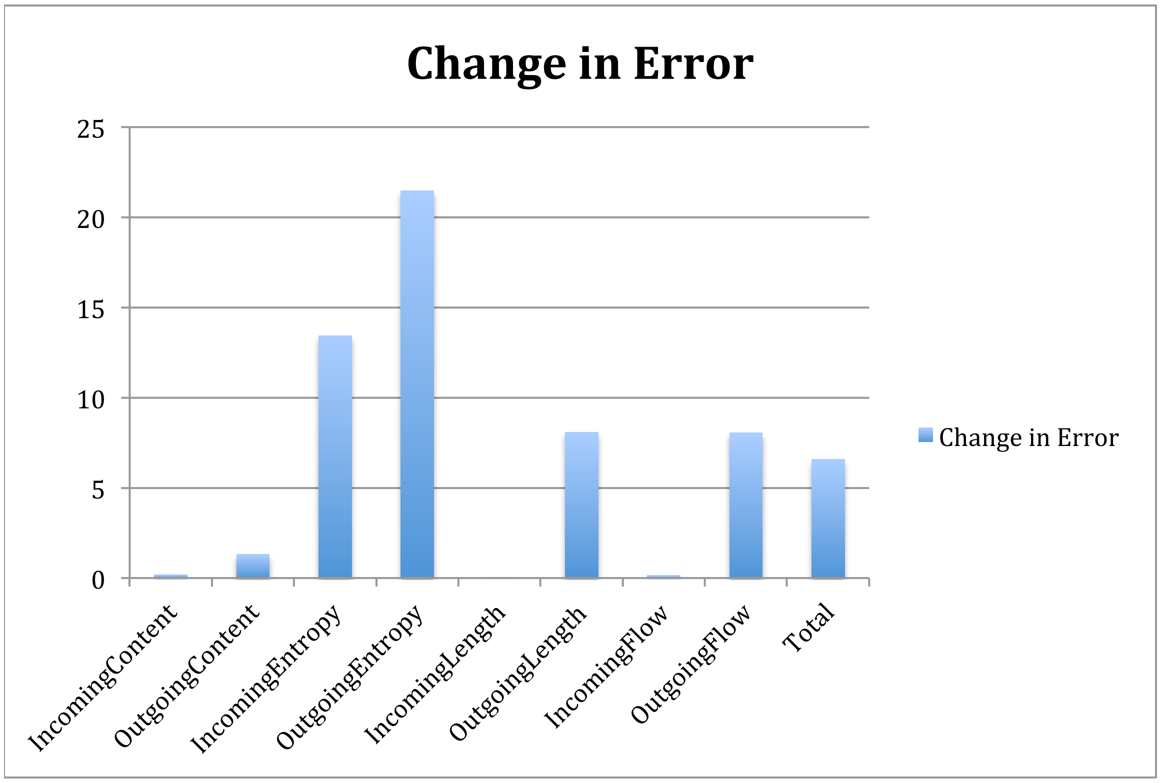

Figure 48: Change in error per feature between XX/YY and XY/YX trials

Figure 48 shows the average change in error per feature that is introduced by switching from XX/YY trials to XY/YX trials. The change in error varies by feature, but is generally low. The change in error is highest for entropy and lowest for content. The change in error is also lower for incoming models than for outgoing models. The average change in error across all features is less than one order of magnitude.

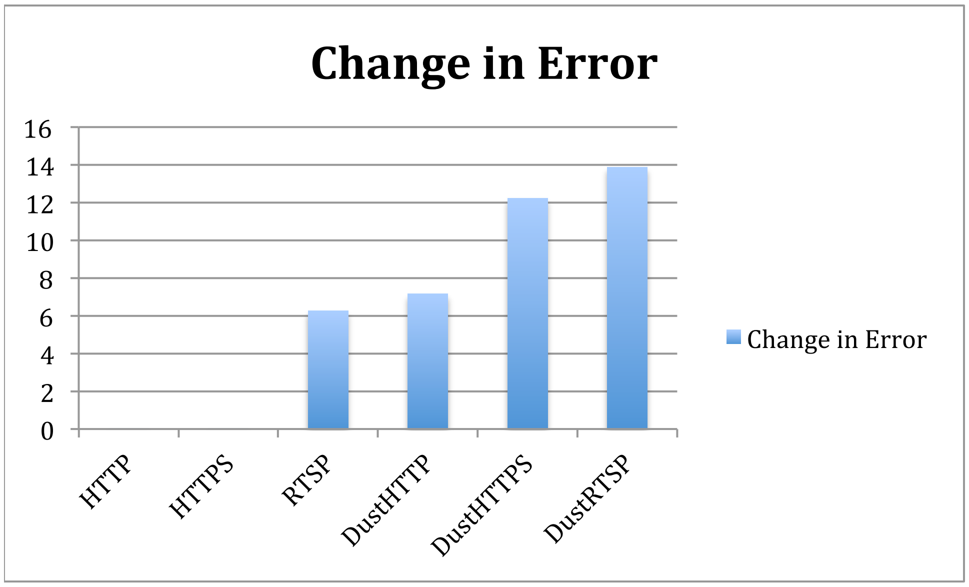

Figure 49: Change in error per protocol between XX/YY and XY/YX trials

Figure 49 shows the change in error per protocol that is introduced by switching from XX/YY trials to XY/YX trials. The change in error varies by protocol, but is generally low. RTSP and the various Dust protocols are more sensitive to changes and training data. This may be because there is more data available for training HTTP and HTTPS models.

The purpose of Figures 48 and 49 is to show that the results are consistent independent of which dataset is using for training and which is used for testing. While the datasets used to change the RMSE scores for the models, the effect is small. Figures 50 through 53 show the average error across all trials.

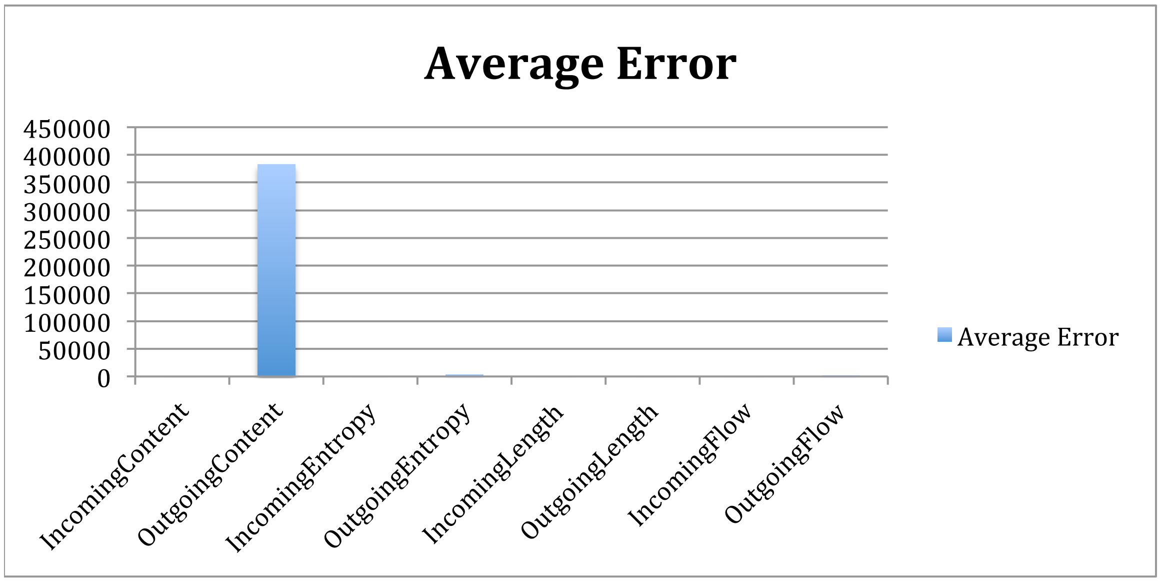

Figure 50: Average error per feature

Figure 50 shows the average error for each feature in both incoming and outgoing directions. The graph shows that by far the most error is incurred on outgoing content.

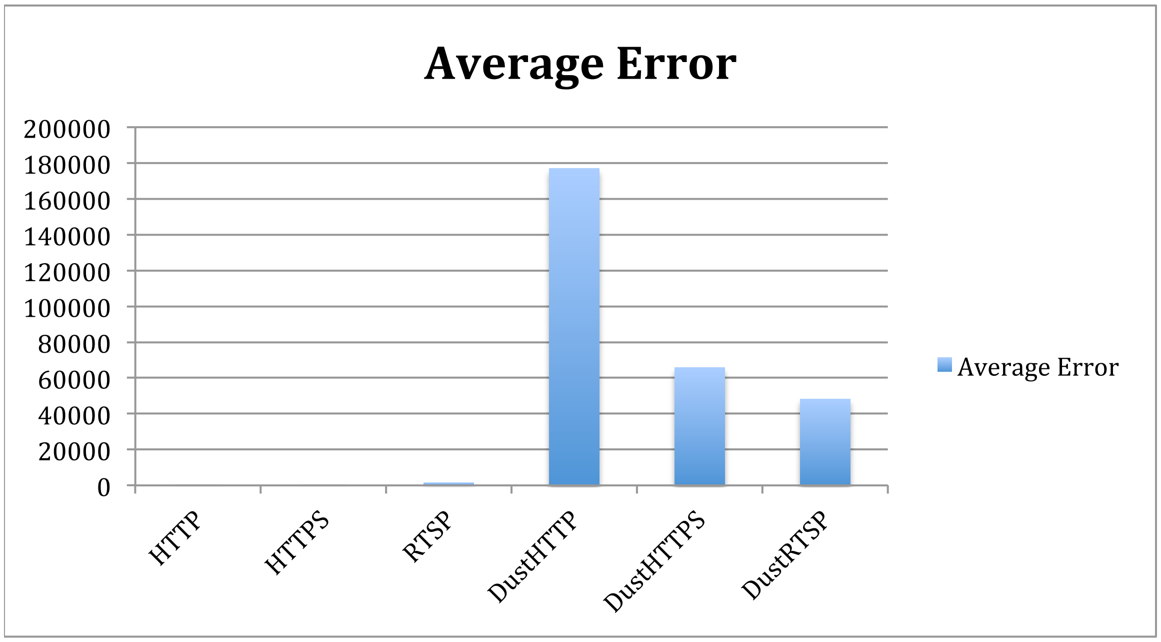

Figure 51: Average error per protocol

Figure 51 shows the average error for each protocol. The graph shows that the average error for the Dust suite of protocols is significantly higher than for the protocols observed in the field. DustHTTP is an order of magnitude worse fit than DustHTTPS and DustRTSP.

Figures 52 and 53 explore more deeply the difference between the goodness of fit for HTTP, HTTPS, and RTSP compared to DustHTTP, DustHTTPS, and DustRTSP.

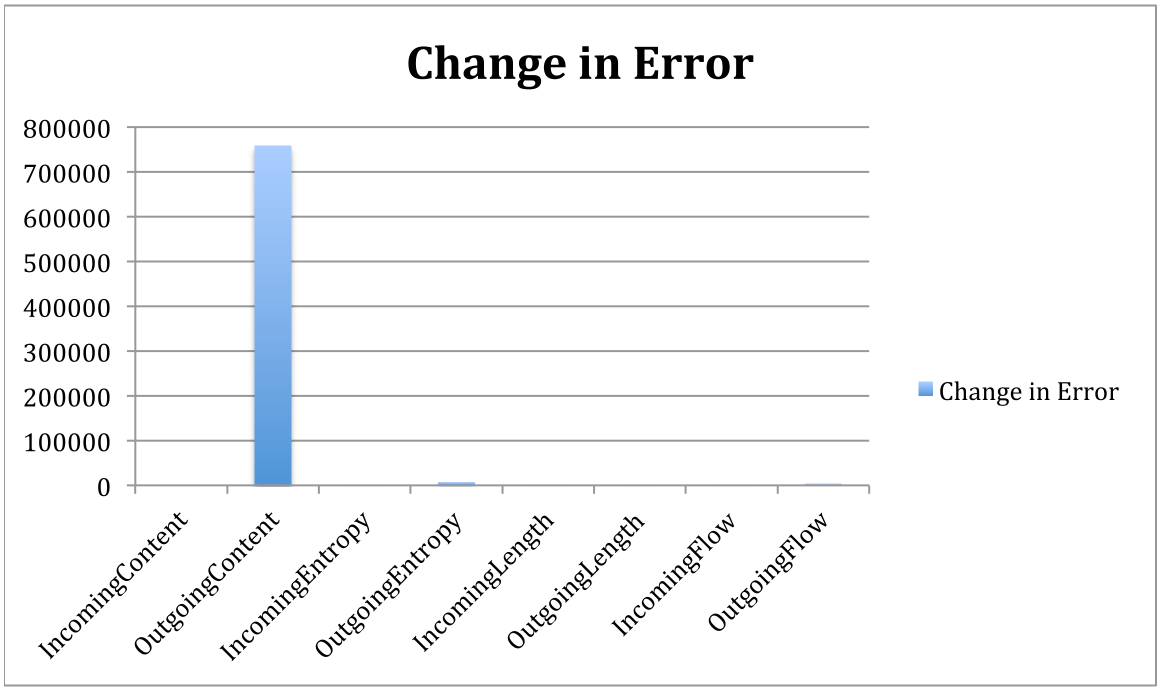

Figure 52: Change in error per feature between original protocol and Dust emulation

Figure 52 shows the change in error that occurs when switching from a protocol such as HTTP to DustHTTP. The graph shows that the most additional error is incurred in the outgoing content feature.

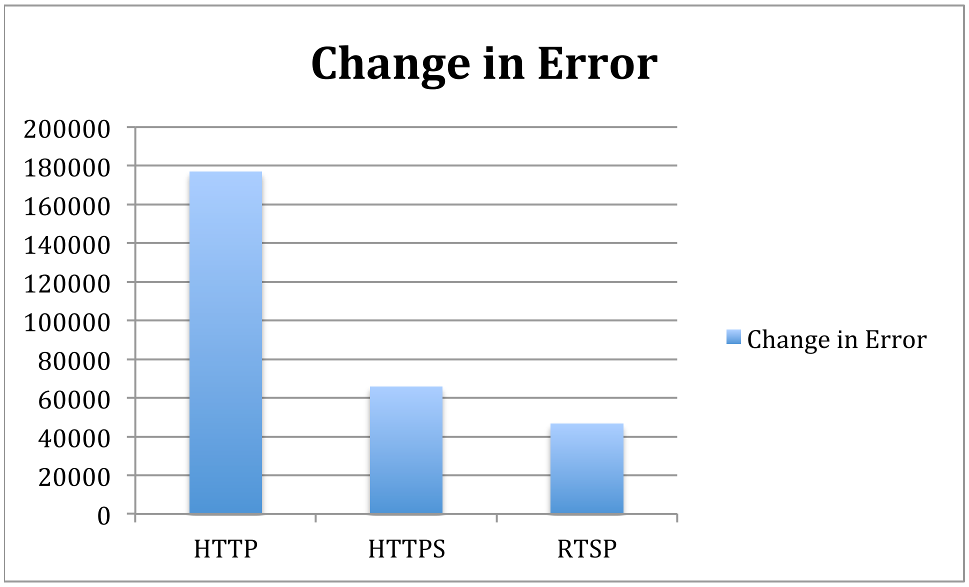

Figure 53: Change in error per feature between original protocol and Dust emulation

Figure 53 shows the change in error that occurs when switching from a protocol such as HTTP to DustHTTP. DustHTTP is the least effective, with an order of magnitude more change in error than DustHTTPS and DustRTSP.

Simulated Adversaries

The full results from doing simulated adversary tests are presented in Appendix B. In these tests, data was again divided into two sets X and Y for each protocol. One set was chosen for training and another for testing. Simulated adversaries use two protocols A and B. For each protocol, there is a training set and a testing set. The data from the testing sets for each protocol were mixed together and then the models derived from the training data for each protocol were used to create a binary classifier that predicted which testing set the data was drawn from, A or B. This classification test could have three possible outputs: positive, negative, or unknown. When compared with the correct answer there were six possible results: true positive, false negative, unknown (but actually from A), false positive, true negative, and unknown (but actually from B). A total accuracy score was calculated by comparing the number of correct answers (true positives and true negatives) with the number of incorrect answers (false positives and false negatives). These total accuracy scores were then classified into categories. Scores where the total accuracy was greater than 90% and the number of unknowns was less than 50% for both unknown categories were considered *Excellent*. Scores with a total accuracy greater than 50% with any number of unknowns were considered *Good*. Scores with a total accuracy less than 50% with any number of unknowns were considered *Bad*. Scores where there were more incorrect predictions than correct predictions for either protocol A or protocol B were considered *Terrible*.

When a simulated adversary had the same protocol for A and B, regardless of dataset, across all features the total accuracy was Terrible. This is the expected result and just serves as a simple sanity check and baseline.

For the HTTP-HTTPS adversary, incoming content was Terrible, outgoing content was Excellent, incoming entropy was Terrible, outgoing entropy was Good, flow in both directions was Terrible, incoming length was Bad, and outgoing length was Good.

For the HTTP-RTSP adversary, content accuracy in both directions was Terrible, incoming entropy was Good, outgoing entropy was Excellent, flow in both directions was Terrible, and length in both directions was Good. These results showed up consistently across X/Y datasets.

For the HTTPS-RTSP adversary, content in both directions was Terrible, entropy in both directions as Good, flow in both directions was Terrible, and length in both directions was Excellent. These results showed up consistently across X/Y datasets.

Overall, the results showed that the results from content were as good as the results from entropy even though entropy is derived from content, with the exception of the outgoing direction on the HTTP-HTTPS classifier, where entropy was still Good but content was Excellent. Additionally, flow was not a good feature to use for classification, whereas length was a good feature to use.

For further discussion, only the features that were Good or Excellent classifiers are discussed, as they are of primary interest, while flow is not discussed further due to its overall poor performance as a classifier. The full breakdown of all results is available in Appendix C.

In the second set of tests, the same adversaries were used, but Dust traffic was substituted for one side. For instance, the HTTP-HTTPS adversary was used to classify a mix of DustHTTP and HTTPS traffic, and separately to classify a mix of HTTP and DustHTTPS traffic. This test was done for all three of the adversaries, substituting Dust traffic for each side separately. Different variations of X and Y datasets were tested. Overall, Dust performed well against the adversaries. The average accuracy for all features was 73% and for all features excepting flow it is even better at 76%.

Finally, a set of tests were constructed in which new adversaries were developed that specifically targeted Dust. For each protocol, an adversary was trained on that protocol as well as the Dust encoding for that protocol. For instance, for HTTP an adversary was developed that was trained on DustHTTP and HTTP data. The same sort of tests were run using the adversary, having it classify a mixture of DustHTTP and HTTP traffic. The purpose of this test was to show the effect of an iterated attacker that is specifically targeting Dust traffic for a particular encoding.

For the DustHTTP-HTTP adversary, content was Terrible, entropy was Excellent, incoming length was either Bad or Terrible depending on the datasets, and outgoing length was Excellent.

For the DustHTTPS-HTTPS adversary, incoming content was Bad, outgoing content was Terrible, incoming entropy was Good, outgoing entropy was Excellent, incoming length was Terrible, and outgoing length was Excellent.

For the DustRTSP-RTSP adversary, incoming content was Terrible, outgoing content was Excellent, entropy was Good, and length was Terrible.

Overall, the adversaries trained on Dust traffic performed well. For HTTP and HTTPS, entropy and length were the best distinguishing features. For RTSP, content and entropy were the best features to use. These tests show that while the current Dust engine and models are good at circumventing filters, they currently do not achieve perfect results and there is therefore room for improvement on the engineering of the Dust engine and the development of better models.

5.3 Conclusion

Overall, the Dust engine performed adequately. The simulated adversaries classified Dust traffic favorably an average of 73% of the time across all features and 76% of the time across all features except for flow. However, new adversaries trained specifically on Dust traffic were effective at distinguishing Dust traffic from the traffic that it mimicked. In one sense, this validates the method of building and training simulated adversaries. In another sense, this points to areas where the Dust engine and the models used can be improved. One particular area of improvement is the entropy reducer. Entropy reduction is accomplished by means of shaping the content distribution. This is accomplished using a reverse Huffman encoder. Unfortunately, Huffman encoding is not a perfect choice for entropy reduction. It quantizes probabilities to powers of two. The probability of each symbol can therefore only be ½, ¼, etc. This works well for distributions that approximate this sort of quantization. However, if the probabilities are very far from powers of two then the quantization can cause artifacts that adversely affect the entropy of the results. As Huffman encoding is being used specifically for achieving a target entropy, this is not optimal. A possible solution is to use a non-quantizing compression algorithm such as arithmetic coding instead.

6. Conclusion

6.1 Summary of Results

The preliminary research began with a study of filtering hardware devices. The results of the hardware study shows that filtering technology in the field is not nearly as advanced as in the computer science literature. The primary methods of classification are based on shallow examination of the packet headers with the primary use of Deep Packet Inspection being matching of byte sequences on a per-packet basis. Entropy was also used by one device to detect and block encrypted protocols. These findings suggest that the priority for encodings should be robust and efficient protection against basic attacks rather than defense against advanced attacks that do not occur in practice.

The second phase of preliminary research was a field study in 4 countries. The packet capture data gathered in this study was used to build a proof of concept HTTP-HTTPS classifier. The results of this research determined that a multinomial distribution would be used for content and packet length, a normal distribution for entropy. An adversary was tuned that could achieve 99% accuracy in classification between HTTP and HTTPS traffic. It was also shown that byte sequence matching is an effective feature for classification.

A value-sensitive design process was then carried out to determine what specific circumvention tool should be created. There were six different design concepts that were explored. What started out as a way to access Twitter eventually ended up as a Dust *Pluggable Transport* for Tor to enable the Courier news reader for Android to be used to access news feeds. This design process also resulted in the selection of three protocols to be used as targets for encoding: HTTP, HTTPS, and RTSP.

The Dust engine was implemented using three layers: the application layer, the encryption layer, and the shaping later. The application layer for Dust as a *Pluggable*

Transport is a single set of frames including chunks of a byte stream to be forwarded. The encryption layer uses a variant of the ntor protocol to establish an encrypted communication channel that is secure against passive observers and active probing attacks. The shaping layer encodes uniformly random encrypted data to follow a target probability distribution provided by the model.

Evaluation of the Dust engine used simulated adversaries that distinguished between HTTP, HTTPS, and RTSP traffic. Overall, the Dust encodings were classified favorably 76% of the time for the features that were selected. A second iteration of adversaries was also done in which they were trained to detect Dust encoded traffic. This second set of adversaries was effective at blocking Dust traffic, leading to interesting avenues for future research in producing more effective encodings.

In revisiting the research questions for the projects, the following answers have been found:

What properties of Internet traffic are important for filtering as it occurs in practice today? The properties of Internet traffic that are important for filtering were identified in the hardware study and later used in the model building and engine design chapter. The important features include packet length, entropy, and byte sequences.

Do the particular filtering methods that have been implemented in deployed hardware have a set of characteristics that provide an opening for a practice of circumvention? Yes, filters use a limited set of features in order to classify network traffic. Circumvention can be achieved by modifying transported traffic so that these features match what the filter will allow through.

How can statistical models be used to capture the relevant details of filters? Models were developed by determining probability distribution functions for each feature. An MCMC method was then used to estimate the parameters for these distributions based on observed data.

How well do the statistical models of protocols fit the observed data? The goodness-of-fit data provided in Chapter 5 shows how well the statistical models fit the observed data across all features and protocols.

How can a circumvention tool be built using models of filters? This process is detailed in Chapter 4. For each feature present in the model, an encoding algorithm was applied to generate traffic conforming to the model representing what the filter would allow through.

How effective is the circumvention tool against the modeled filters? An evaluation is presented in Chapter rap5. To sum up the overall performance in a single number, the percentage of favorable classifications when traffic encoded using the Dust engine was classified by the simulated adversaries was 76%. More detailed information, including all of the different tests run with different features and data sets is available in Appendix C.

What are the characteristics of traffic carrying credible and relevant information that allow it to be classified and filtered? In the final design concept that resulted from the value-sensitive design process, the source of credible and relevant information was RSS news feeds, which are transferred using the HTTP protocol. As shown in Chapter 5, HTTP can be effectively classified by content, entropy, and packet length.

How effective can a circumvention tool be in restoring access to this information when evaluated against a simulated filter? In the final design concept that resulted from the value-sensitive design process, the circumvention tool was a *Pluggable Transport* for Tor. The goal of this tool is to be deployed with the Courier news reader for Android. However, due to the generic nature of *Pluggable Transports*, it can also be used to transport network traffic for other applications that are integrated with either Tor or directly with the *Pluggable Transports* framework. This opens up the possibility, for instance, of reading news by accessing a news website through a web browser, rather than using a news reader application. As is shown in Chapter 5, overall the predicted effectives of the tool in circumventing network filtering for the simulated adversaries was 76%.

How efficient can a circumvention tool be in restoring access to this information in terms of bandwidth overhead? In the final design concept that resulted from the value-sensitive design process, the circumvention tool was a *Pluggable Transport* for Tor. Tor has bandwidth requirements that are much greater than those of RSS news feeds. At this point in the design process, the question of bandwidth overhead was no longer a straightforward comparison. Instead, the technical challenge was to provide sufficient performance for Tor to function. This was achieved, as all tests of the Dust engine were completed using a functioning Tor network connection. Therefore, bandwidth overhead was low enough to enable Tor to function and for resources to be fetched over HTTP.

6.2 LIMITATIONS

In an ideal world, models could be built from perfect information with no missing data. Network analysis could take place on every network and not just in the 4 countries from the field study. Every filter device could be studied individually rather than just the two used in the hardware study. When developing a circumvention tool, the filter model would be based on complete information about the filtering conditions on the target network. One of the limitations of this study is that such perfect information is not available. It is difficult to obtain filtering hardware, and field testing is limited to countries where research collaborators can be found. These are limitations that face all unobservable communication researchers. Fortunately, Bayesian statistical models are well suited to dealing with incomplete information. Where information is missing about a specific filter, it can be inferred from the observed data. While there is always the possibility of outliers that do not fit the models derived from observable data, Bayesian models improve with the amount of available data. All observed instances of filtering are potential data for the model, so even a circumvention tool that fails to be effective for a particular adversary provides information for improving the model.

Another potential limitation is that Bayesian statistical models may not be capable of modeling all possible filters with sufficient accuracy to be effective. A method of filtering could be hypothesized that is resistant to representation using a statistical model. For instance, a filtering algorithm based entirely on constraints, such as one in which the sole criteria for classification is whether or not the message is well-formed with respect to a formal grammar, would require a semantic rather than statistical model. An example of such a filter would be one that requires all HTTP traffic to be valid HTML. A statistical model makes a poor approximation for these sorts of rules, and a semantic model such as a formal grammar would be a better fit. There are some arguments that support the idea that this limitation is not problematic in practice. First, the preliminary research shows that filters use classification algorithms that can be modeled statistically. Second, the Dust engine actually uses a hybrid semantic-statistical approach. The focus is on statistical models because they bear the most relevance to the techniques used in deployed filters. However, semantic constraints can be implemented as well if they are existential in nature. Including required byte sequences is an example of an existential semantic constraint that is currently supported by the Dust engine. Finally, if there are outliers that use advanced semantic techniques for classification, then they are relatively rare in terms of deployment. Focusing on the filters that are being used to block access to credible and relevant information to specific user communities means that these advanced but rare devices have less impact on this research than they would on a research program focusing on worst-case scenarios for circumvention tools.

In additional to the above limitations, there are also some limitations on the current implementation that are practical compromises and diverge from the ideal realization of the core concepts. It was determined during the value-sensitive design process that perfect forward secrecy (PFS) was a necessary feature in order for Dust to be taken seriously by the security community. However, there is a trade-off in adding this feature inasmuch as PFS requires at least one round trip before any encrypted communication can occur. This limits the scope of possible transports that can be used to send information. For instance, HTTP is not a suitable transport because HTTP connections have a single request followed by a single response, resulting in only one round trip. By the time the round trip necessary to establish PFS is complete, the HTTP connection is closed before any information could be sent. Therefore use of a transport such as HTTP would require a different message architecture than the one used by Dust. In Dust, each TCP connection must perform its own encryption handshake, whereas to use HTTP it would be necessary to use the first HTTP connection to set up the connection and subsequent connections to send information. Additionally, the specific use case of Dust as a *Pluggable Transport* is one where features such as PFS might not be so important. Dust as a Pluggable Transport wraps the Tor protocol and Tor could provide PFS instead.

Another limitation is on the minimum size of transmission. For a TCP connection to be used to send information, it must be long enough in terms of duration, flow, and packet size to complete a full handshake followed by at least one data frame. Connections that are too short only provide filler and can never send actual information. If there are too many connections of this short variety (for instance, all of them), then the rate of data exchanged is too low to be useful. Therefore, the range of possible models is limited by this bound. Ideally, this would not be the case and Dust would be able to use any model whatsoever. Achieving this goal would require reconsidering the design requirements for the protocol to make the right trade-offs between required security properties and the limitations placed on the possible models by these requirements. In the use case of Dust as a Pluggable Transport, this particular limitation is overshadowed by a larger limitation of the Pluggable Transport subsystem. It assumes one transport connection for each application connection. Therefore, although the Dust engine was originally designed to split application traffic over multiple transport connections, this feature cannot be utilized in the current Pluggable Transport framework. Therefore, the duration feature of models is currently ignored. This is something that should be revisited in future work because duration is a feature that can be used to successfully differentiate Tor traffic from various encodings such as HTTP and HTTPS. However, this is something that must be changed in the Pluggable Transport framework first before Dust as a Pluggable Transport can make use of duration shaping capabilities.

6.3 IMPLICATIONS

From the multiple iterations of design, some themes stand out. First, there are essentially two classes of stakeholders in the world of circumvention tools. There are the actual users of the tools and there is the larger circumvention community, which includes developers, funders, enthusiasts, journalists, and others. Some members of the larger community are users of the tools and some are not. These two sets of stakeholders have somewhat different needs, but to some extent both must be satisfied with the same tool. If the community does not accept the tool, adoption among end users is difficult. However, writing tools for the larger community alone ignores the needs of the actual users of the tools. Blending the needs of these two groups is a significant design challenge. A high-level comparison of the two groups suggests that end users want tools that "just work" with a minimal of change to existing behaviors and without the need to learn new concepts and interfaces. What users want is essentially the existing Internet, but without filtering, delivered over a filtered network. The circumvention community, however, has more complex needs. They value privacy and security in their circumvention tools as much as the actual circumvention of filtering. Open source software and cryptographic protocols verified by professional cryptographers are high priorities. Proprietary commercial solutions are viewed with distrust. The downside of tailoring solutions to this community is that the tools are often difficult for end users to use. The emphasis on the perfection of the privacy and security aspects of the software, aspects that end users do not seem to place a high priority on, take development effort away from creating a simple and intuitive interface that "just works".

It is an adage in engineering that you can't optimize for everything at once. However, the needs of the end users can't be optimized to the point that the circumvention community rejects the tool. A proposed solution to this problem that was implemented in the Dust design is to optimize for security and privacy only when it does not require additional interaction with the user. The user interface must be able to be developed orthogonally to the security and privacy decisions so that it can be optimized for ease of use. An example of this problem and the proposed solution in the Dust engine is the cryptographic handshake. Dust uses a cryptographic handshake in order for the client and server to exchange an encryption key so that they can encrypt and decrypt the communications between them. In Dust, this encryption is not meant to provide any security or privacy. It is only meant to randomize the content in the encryption layer so that it is uniformly random before it is fed into the shaping layer. Uniformly random content ensures that no characteristics of the distribution of the original content are represented in the shaped content. Dust therefore used a very simple cryptographic handshake. However, the refrain from the circumvention community has been that a feature called “perfect forward secrecy” is required. What this feature entails, essentially, is that the client and server use keys in the handshake that they then discard. This ensures that if the client or server is compromised after the communication takes place (and the adversary has a recording of the encryption communication), the adversary is unable to recover the keys necessary to decrypt the communication after the fact. This attack makes little sense in the context of Dust as the encrypted message is only intended to stay secret just long enough to get past the filter. If the adversary discovers that the client and server were communicating with Dust, there is no need to attempt to decrypt the message. The adversary can simply add the server’s IP to the blacklist and further communication is not possible with that server. Ensuring perfect forward secrecy in the Dust key exchange requires doing a two-round key exchange instead of a single round. This increases the complexity of the key exchange and therefore the code that implements it. It also restricts the ways that Dust can communicate. For instance, messages cannot be sent unidirectionally as the key exchange is bidirectional. However, since no changes to the user interface were required, perfect forward secrecy was implemented. Overall, every attempt was made to conform to the standards of practice for the circumvention community, whether or not they made sense in this context, as long as they did not impact interaction with the user.

Finally, the ways in which users obtain access to credible and relevant information appear to be regionally and culturally specific. This project started with an American point of view that assumed social media was the primary source for credible and relevant information for people around the world. Further research has shown that there is a surprising diversity of information sources. The reasons for this are possibly a complex mixture of cultural, economic, and political factors. Fortunately, the most straightforward way to determine which medium users rely on for credible and relevant information is to ask the users.

The implications of this research are changes to local filtering conditions for the target user communities, as well to the global landscape of filtering devices and circumvention tools. Thus far the landscape of filtering devices and circumvention tools has been one of cycles of competing products. Network operators choose one of a number of off-the-shelf products with limited configurability in order to meet the filtering mandates that they are given. These devices primarily enable network operators to turn on or off filtering based on a taxonomy of high-level filtering categories. They provide problematic ontologies in which there is no transparency or accountability into how these categories are defined. The fundamental control over filtering decisions is made by the device manufacturers when they construct these ontologies and not by the network operators or the users of the networks. Users that are negatively impacted by the deployment of filtering devices in their everyday information-seeking behavior have a similar choice of a number of off-the-shelf circumvention tools. These tools are designed to be one-size-fits-all and are not generally customizable for the filtering conditions on specific networks. Changes in the configuration of filtering devices prompts new versions of the circumvention tools. Revised circumvention tools prompt changes in the configuration of the filtering devices. Each side is attempting to create general solutions where filters must defeat all circumvention tools and circumvention tools must defeat all filters.

The Dust approach to building circumvention tools disrupts this cycle of revising the filtering and circumvention tools. With every change to the configuration of the filters, a new model can automatically be built from the observed changes in filtering conditions. This new model can be substituted for the old model without the need to revise the circumvention tools. This allows for faster development and deployment as well as increased customization. Rather than having one obfuscated protocol in use on all networks, a custom encoding can be developed for each network based on filtering conditions. This proliferation of encodings cannot be addressed by the manufacturers of filtering devices simply by adding a few new categories - as has been done in the past with conventional obfuscated protocols. Rather than adding a new protocol to the network, the Dust engine adds a family of protocols derived from a large space of possibilities. Dealing with this in a systematic way in the filtering hardware requires a paradigm shift in the way filters are constructed. Simply reconfiguring a device will not be sufficient. Instead, a new line of products must be developed, sold to network operators, and deployed to replace the old devices that classified protocols based on static properties. Common optimizations for scaling, such as only looking at the first packet of a conversation, will no longer be viable. Instead, what will be required are new methods for efficiently doing more sophisticated analysis without adversely affecting bandwidth or latency beyond acceptable limits.

In the short term, users will benefit from regaining access to credible and relevant information while the new filtering hardware is being developed and the network infrastructure is being upgraded. With the current product upgrade cycles for national-level network infrastructure, the short term could be measured in years.

Long-term monitoring of traffic to find statistical anomalies will require long-term data storage and skilled analysts to look for anomalous patterns, increasing both the number of devices needed to filter the same amount of traffic and the staff required to operate the devices. In the long term, the dialectic between filtering and circumvention moves from one of technology to one of information. In a contest between a filter and a circumvention tool, the one with the better model for how to accurately classify traffic will win. The long-term outcome will therefore depend on which is easier to do, to encode blocked traffic so that it resembles allowed traffic or to write a detection algorithm that can differentiate encoded traffic from allowed traffic. While this research does not claim to be an end to network filtering, it advances the state of circumvention to the point that current filtering methods are no loner viable, requiring a paradigm shift in the design of filters.

Another issue to consider when discussing the implications of this research is the possibility of potential negative consequences of this research, such as misuse, collateral damage, and unintended effects. This is an issue that faces the development of all technological interventions. Circumvention research commonly takes a neutral approach regarding unintended consequences. However, a value-sensitive-design perspective necessitates deeper consideration of the potential for negative impact. The most direct negative impact is that of intentional misuse of the technology for unintended uses. It is often the case that computer security research intended for defensive purposes can also be used for offensive purposes. In the case of this work, no specifically offensive uses are obvious. The techniques presented here do not infiltrate, disable, damage, or destroy filtering hardware. The Dust engine essentially provides a reliable communications channel in a hostile environment and therefore serves more of a utility function similar to radios, telephones, and the Internet.

Of course communication channels of all sorts can be used for a variety of purposes, both positive and negative, including criminal activities and the coordination of activities, contrary to the project's stated core value of providing access to credible and relevant information. Addressing these issues as they relate to filtering circumvention technology is a matter of examining how filtering technology is used. Some filtering is used to curtail criminal activity, specifically by shutting down illegal websites. In some cases, circumvention technology has been used to bypass this filtering, thereby enabling criminal activity. However, blocking specific websites is a targeted attack. Filtering circumvention tools have only been able to provide at best a temporary relief from targeted attacks. When specific websites are targeted, the servers that run them can eventually be found and shut down and the server operators arrested. Even sophisticated designs such as Tor are vulnerable to long-term intersection attacks on specific users [23]. The current practice of protocol-based filtering, however, is a wide net, blocking all users of a protocol regardless of whether their use is legitimate or criminal. It is this type of filtering that Dust prevents and not targeted attacks against specific users. Therefore there is not a high risk of its appropriation for criminal activities as the problem it solves is one that is more salient for everyday information seekers than for criminal organizations.

Regarding collateral damage, a likely effect if adoption of Dust technology becomes widespread would be reconfiguration of the filters to attempt to block Dust traffic. This could possibly result in collateral damage in the form of an increase in the false positive rate of the filter, causing additional traffic to be blocked that was not blocked before. This sort of escalation of filtering conditions as a response to circumvention has been seen previously. For instance, switching a website where individual pages are being selectively filtered from HTTP to HTTPS defeats the selective blocking of pages and can case the whole site to be blocked [19]. However, since Dust is adaptive, configuring filtering settings will only provide a temporary solution. If the Dust design is successful, then repeated reconfiguration will eventually result in blocking all traffic, effectively disconnecting the network. However, widespread deployment of Dust is outside of the scope of this project. The circumvention tools being designed are intended for small-scale deployment that would be unlikely to result in reconfiguration of the filtering hardware.

6.4 Future Work

In terms of model building, the present work is the starting point for what could be a lifetime of research into building models to simulate the characteristics of network traffic. Future research into model building could explore three categories of variation from this starting point: more complex models, additional machine learning methods, and larger datasets. The models presented here are the simplest models that were found to achieve adequate performance. Therefore, simple distribution functions were used such as multinomial and exponential distributions. More complex mixtures of multiple distributions and hierarchal models where the distribution parameters are also distributions rather than scalars are also possible. More complex models offer additional challenges inasmuch as adding complexity can cause a model to overfit the data, resulting in excellent goodness-of-fit scores for the training data but poor performance in predicting outcomes for other datasets. Therefore, alternative metrics should be considered for evaluating more complex models that balance complexity with goodness-of-fit. This work has used a Bayesian method of distribution parameter estimation by means of the Markov Chain Monte Carlo (MCMC) method. This is just one method from a diverse field of machine learning algorithms used for classification problems. Other methods such as support vector machines (SVM) are often employed for similar feature-based categorical classification tasks. This rich field of study could be mined for a diverse array of techniques for classifying network traffic. Similarly, the byte sequence extraction technique could possibly improved upon using any of the numerous techniques from the field of text data mining. While there are a number of different modeling and machine learning techniques to draw from to extend and expand this work, significant progress in building higher performance models may be obtained using the present techniques with larger datasets. This dissertation used data for HTTP and HTTPS from a multi-country field study. While this is a large quantity of collected data relative to other projects in the field, it is a small sample of the overall volume of network traffic on the Internet. Data collection could be extended in several dimensions: more countries, more protocols, more sites visited for each protocol, more instances of data collection, and collection over greater time periods and geographic areas within each country.

Another area where more research could be done is in the determination of final classification scores through the combination of scores from the separate features. In the current implementation, features are considered to be independent and the scores from each feature are added. Each feature gets one "vote" and the classification with the most votes wins. An alternative means of scoring would be to incorporate the confidence measure implicit in each feature score. If the scores across classifications for one feature vary widely, this indicates a higher confidence than if the scores are very close. However, scores cannot be compared across features because they fall in different ranges. In order to incorporate this information into the final score, the scores would need to be normalized to a common range across all features. A simple way to do this would be to examine the variance and standard deviation for each feature and normalize scores accordingly. However, there are also a variety of more complex techniques. For instance, principal component analysis (PCA) can be used to provide several enhancements, such as normalizing the scores, reducing the dimensionality, and exposing the covariance between features that were previously assumed to be independent. There is a diverse space of scoring methods that could be researched to provide boosts in the accuracy of the adversaries.

This research represents a significant advance in the state of the art for circumvention technology. It also reveals the limitations of the current methods and tools used for circumventing Internet filtering. In future work, the capabilities of the tools will be expanded to allow for further research. Most obfuscating protocols currently use a single IPv4 TCP connection. As a result, protocols such as HTTP and HTTPS have been studied extensively. More capable tools would allow for exploration of transports that use UDP, IPv6, and multiple TCP connections. With multiple TCP connections, both splitting up traffic over simultaneous parallel connections and over a series of connections could be explored. This advancement would also allow for transporting tunneled connections over a mixture of different transport connections. For instance, a mix of different protocols could be used, or even a mixture of TCP connections and UDP packets. This use of ensemble connections opens possibilities for new types of modeling. In addition to modeling the properties of individual connections, the properties of mixtures could be modeled. Properties such as the length of connections and the probability and timing of different types of traffic could be modeled in order to synthetically reproduce a mixture of traffic resembling “normal” Internet traffic. Such a research program would help elucidate the open question of what “normal” Internet traffic looks like. Answering this question will require field work to observe and sample network traffic to establish a baseline.

In addition to expanding the capabilities of the tools in general, the capabilities of Dust can also be extended and optimized for efficiency. Currently, Dust always shapes all properties for maximum obfuscation. However, shaping each property adds a performance penalty, and not all properties are actually used for filtering by all filters. A modular Dust engine that shapes only relevant properties could have significant performance gains, particularly when packet timing can remain unshaped. The models used by Dust could also be optimized. Currently, the simplest possible models are used for each property, consisting of a single probability distribution. More complex models such as mixture models could provide better fit to the target empirical distribution. The space for exploration here is open ended. In addition to Bayesian modeling, other methods of statistical modeling such as machine learning methods could be used to generate target distributions. How to find and identify the optimal distributions for each property is an open research question. Another open research question is how to optimally encode to match the target distribution for each property. For instance, in the case of the content distribution Dust currently uses Huffman encoding. Other methods such as arithmetic coding could be used instead. The literature on coding theory and data compression contains several methods that could be tried and compared in the search for an optimal encoder. There is similar research that could be done on all of the properties.

The design and evaluation of circumvention tools using Dust is at the beginning of a long program of research into the pragmatic dimension of circumvention tools. The current research has undergone numerous revisions and explored the intersection between the needs of users and the capabilities of the current circumvention tool ecosystem. As the capabilities of tools are expanded, this will open up further opportunities for value-sensitive design research. The most significant design compromise motivated by technical limitations rather than user needs came from the bandwidth requirements necessary to support Tor traffic. This was a necessary compromise as integration with Tor is the dominant method of deploying new obfuscating protocols. However, it also constrains obfuscation research in terms of the optimal efficiency that is possible. There is a possible research opportunity in developing obfuscation techniques for lower bandwidth and higher latency applications than what is possible when integrated into Tor. As Tor has developed its internal obfuscation technology in the form of *Pluggable Transports*, there is a possibility that other projects with different bandwidth and latency requirements will also utilize these transports. If so, and if users adopt these other systems, then that would provide an opportunity for further value-sensitive design research on obfuscating transports for circumvention tools with a different set of requirements and constraints.

Appendix A – Field Study Data

The data for the field study has been rendered as a series of charts that can be obtained as a separate PDF file at http://blanu.net/AppendixA.pdf.

Appendix B – Full Size Bytes Sequence Images

Full size images of the byte sequence analysis results can be obtained as a separate PDF file at http://blanu.net/AppendixB.pdf.

Appendix C – Tables of Evaluation Results

Complete tables for the evaluation results can be obtained as a separate PDF file at http://blanu.net/AppendixC.pdf.

Appendix D – Source Code for Software

All of the tools developed for use in the course of this research are open source and available for use by other researchers under a permissive license. The software can be downloaded by the Github source code hosting website. The source repositories for the tools can be found at the following locations:

- Dust-tools – https://github.com/blanu/Dust-tools
- CensorProbe – https://github.com/blanu/CensorProbe
- Adversary Lab – https://github.com/blanu/AdversaryLab-offline
- Dust – https://github.com/blanu/Dust
- Dust patch for obfs4proxy – https://github.com/blanu/obfs4

The Dust-tools package is discussed in greater detail in Appendix E.

Appendix E – Hardware Study Testing Tools

For the hardware study, a set of tools was developed for testing the capabilities of filtering hardware. These tools were released as open source under the name "Dust-tools". Information for downloading the software created in the course of this dissertation, including Dust-tools, is available in Appendix D. There are two basic tools in the package: Replay and Shaper. Each is discussed in turn in this section, as well as additional utilities useful for debugging.

REPLAY

Replay takes a recorded conversation and replays it, possibly with variations. This is used to detect bytestring matching in the filter. By varying the content of the recorded conversation, bytestrings used by the filter can be detected. The Replay tool requires a recorded conversation in the form of a Packetstream file. Packetstream files are a cross-platform format which can be obtained by converting .pcap files. The .pcap files can be recorded using a packet capture tool such as Wireshark, tcpdump, or CensorProbe. Converting .pcap fils to the Packetstream format requires the Dust-tools-pcap software package (included in the main Dust-tools code repository), which only works on Linux. However, the Replay tool is cross-platform.

The replay client and server can take an optional mask file which specifies variations. The mask file is in the following format:

```
offset,byte offset,byte offset,byte ...
offset,byte offset,byte offset,byte ...
...
```

Each line maps to a packet in the replayed conversation. The first line corresponds to the first packet and so on. Each line contains a sequence of offset and byte pairs. The offset specifies the index within the packet to modify. The byte specifies the value to set for that offset. For instance, 0,32 would set the first byte (index 0) of the given packet to the value 20, corresponding to an ASCII space.

Example Mask File

```
 0,0 1,0 2,0
0,32 7,32 12,32
```

The above sets the bytes at the 0, 1, and 2 positions of the first packet to 0 (null) and the 0, 7, and 12 positions of the second packet to 32 (space). The usefulness of the replay tool is in detecting bytestrings used by filters to classify traffic. For instance, HTTP traffic starts with a byte sequence such as, “GET /favicon.ico HTTP/1.0”. Perhaps the filter is looking for "GET" at the beginning of the first packet to identify HTTP traffic. We can test this hypothesis by creating a mask file such as, “0,0 1,0 2,0”. This sets the first three bytes of the first packet to 0 (null). We then use the replay tool to replay the HTTP traffic, but with those three bytes set to 0. If we were to view this replayed traffic it would look like, “??? /favicon.ico HTTP/1.0”. The question marks represent null values that cannot be printed. If the filter fails to classify this traffic as HTTP then the hypothesis that it is matching "GET" at the beginning of the first packet is confirmed.

SHAPER

Shaper takes a protocol model and generates random traffic that conforms to that model. This is used to determine if the protocol models are accurate with respect to a specific filter. If the filter treats actual traffic and Shaper traffic the same, then the model is accurate with respect to that filter. For instance, if the filter blocks both actual and Shaper HTTP traffic, and passes through both actual and Shaper HTTPS traffic, then the HTTP and HTTPS models capture the characteristics that the filter is using to distinguish HTTP traffic from HTTPS traffic.

The Shaper tool requires a protocol model in the form of an Observation file. Observation files are a cross-platform format which can be obtained by converting .pcap files or using live traffic capture. The .pcap file recording step can be skipped by capturing live traffic directly to Observation files. Both converting .pcap files and recording live traffic require the Dust-tools-pcap package (included in the main Dust-tools code repository), which only works on Linux. However, the Shaper tool is cross-platform.

The shaper-update utility is used to capture live traffic and convert .pcap files into Observations and can be run multiple times on the same Observation file to aggregate observed traffic. Once the Observation file has been prepared, the Shaper tool can be used to generate traffic. The Shaper consists of a client and a server which are both given the same Observation file. The client and server will exchange traffic which conforms to the model specified in the Observation file forever until interrupted with Ctrl-C. This can use a lot of bandwidth, so they should only be run for a short time and then stopped. The accuracy of the models can be determined by looking at how the filter treats the Shaper traffic as compared to the actual traffic from which the models were derived.

ADDITIONAL UTILITIES

Dust-tools contains additional utilities for working with Observation files. The shaper-show utility displays the contents of an Observation file. This can be useful for debugging models. The shaper-export utility converts the contents of an observation file to a CSV file. This can be useful for making graphs of models.

References

1. Bittau, A., Hamburg, M., Handley, M., Mazieres, D., and Boneh, D. The case for ubiquitous transport-level encryption. *19th USENIX Security Symposium.*, (2008).

2. Boesgaard, C. *Unlinkability and Redundancy in Anonymous Publication Systems*. Denmark, 2004.

3. Cisco. Using NetFlow Sampling to Select the Network Traffic to Track. In *Cisco IOS XE NetFlow Configuration Guide*. San Jose, CA, 2009.

4. Clarke, I., Sandberg, O., Wiley, B., and Hong, T.W. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Designing Privacy Enhancing Technologies*. Springer-Verlag, Berlin, 2001, 46–66.

5. Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. *In: Proc. of the 13th USENIX Security Symposium*, (2004).

6. Dingledine, R. and Mathewson, N. Design of a blocking-resistant anonymity system Tor Project technical report , Nov 2006. *Design*, (2006), 1–24.

7. Dingledine, R. Tor and circumvention : Lessons learned. *The 26th Chaos Communication Congress*, (2009).

8. Dyer, K.P., Coull, S.E., Ristenpart, T., and Shrimpton, T. Format-Transforming Encryption : More than Meets the DPI. .

9. Dyer, K.P., Coull, S.E., Ristenpart, T., and Shrimpton, T. Protocol Misidentification Made Easy with Format-Transforming Encryption Categories and Subject Descriptors. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, (2013).

10. Friedman, B., Howe, D.C., and Felten, E. Informed Consent in the Mozilla Browser: Implementing Value-Sensitive Design. *Proceedings of the 35th Hawaii International Conference on System Sciences*, IEEE Computer Society (2002), 247.

11. Friedman, B. Value-sensitive design. *Interactions 3*, 6 (1996), 16–23.

12. Friedman, B., Denning, T., & Kohno, T. Security Cards: A Security Threat Brainstorming Toolkit. 2013. http://securitycards.cs.washington.edu/.

13. Granerud, A.O. Identifying TLS abnormalities in Tor. *Information Security*, (2010).

14. Hand, S. and Roscoe, T. Mnemosyne: Peer-to-Peer Steganographic Storage. *IPTPS*, Springer-Verlag (2002), 130–140.

15. Harrison, D., Ciani, A., Norberg, A., and Hazel, G. Tracker Peer Obfuscation. 2008, 1–8. http://bittorrent.org/beps/bep_0008.html.

16. Hevia, A. and Micciancio, D. An Indistinguishability-Based Characterization of Anonymous Channels. *PETS*, Springer-Verlag (2008), 24–43.

17. Hjelmvik, E. *Breaking and Improving Protocol Obfuscation*. 2010.

18. Houmansadr, A., Brubaker, C., and Shmatikov, V. The Parrot Is Dead: Observing Unobservable Network Communications. *2013 IEEE Symposium on Security and Privacy*, (2013), 65–79.

19. Hunter, P. Pakistan YouTube block exposes fundamental Internet Concern that Pakistani action affected YouTube access elsewhere in. *Computer Fraud & Security 2008*, 4 (2008), 10–11.

20. Kadianakis, G. *obfs2*. 2013.

21. Kopsell, S. and Hillig, U. How to achieve blocking resistance for existing systems enabling anonymous web surfing. *Privacy in the electronic society*, ACM Press (2004), 47.

22. Leidl, B. Obfuscated-OpenSSH README. 2009, 1–6. https://github.com/brl/obfuscated-openssh/blob/master/README.obfuscation.

23. Øverlier, L. and Syverson, P. Locating Hidden Servers. *2006 IEEE Symposium on Security and Privacy*, (2006).

24. Perng, G., Reiter, M.K., and Wang, C. Censorship Resistance Revisited. *IH*, Springer-Verlag (2005), 62–76.

25. Pfitzmann, A. and Kohntopp, M. Anonymity, Unobservability, and Pseudonymity – A Proposal for Terminology. *Anonymity*, Springer-Verlag (2001), 1–9.

26. Sennhauser, M. The state of iranian communication. 2009. http://emsenn.com/wp-content/uploads/2009/07/SoIC-1.21.pdf.

27. Sennhauser, M. The state of iranian communication. *Response*, (2009).

28. Serjantov, A. Anonymizing censorship resistant systems. *IPTPS*, Springer-Verlag (2002), 111–120.

29. Topolsky, R.M. *In the Matter of the Petition of Free Press et al. for Declaratory Ruling that Degrading an Internet Application Violates the FCC's Internet Policy Statement and Does Not Meet an Exception for "Reasonable Network Management."* 2007.

30. Waldman, M. and Mazieres, D. Tangler: a censorship-resistant publishing system based on document entanglements. *Computer and Communications Security*, ACM Press (2001), 126–135.

31. Waldman, M., Rubin, A.D., and Cranor, L.F. Publius: A robust, tamper-evident, censorship-resistant web publishing system *. *9th USENIX Security Symposium*, (2000).

32. Message Stream Encryption (aka PHE) format specification. 2006, 1–7. http://wiki.vuze.com/w/Message_Stream_Encryption.

33. Obfuscated TCP. 2010. http://en.wikipedia.org/wiki/Obfuscated_TCP.

34. *Internet in Chains: The Front Line of State Repression in Iran*. 2014.

Vita

Brandon has worked for the last ten years in open source and peer-to-peer software, both in community projects and tech startups. He founded a number of open source peer-to-peer software projects, including Freenet, Tristero, Alluvium, and Project Snakebite. He has also worked in peer-to-peer Interent video delivery at Swarmcast as Senior Engineer and then at BitTorrent as the Director of Product Management.

Email: brandon@blanu.net

This dissertation was typed by Brandon Keith Wiley.